論文 / 著書情報
Article / Book Information

題目(和文)	DEMOにもとづく情報システム開発
Title(English)	DEMO-based Information Systems Development
著者(和文)	ターリク・ファタヤーニ
Author(English)	Tarek Fatyani
出典(和文)	学位:博士(学術), 学位授与機関:東京工業大学, 報告番号:甲第10281号, 授与年月日:2016年6月30日, 学位の種別:課程博士, 審査員:飯島 淳一,妹尾 大,永田 京子,中田 和秀,佐伯 元司
Citation(English)	Degree:Doctor (Academic), Conferring organization: Tokyo Institute of Technology, Report number:甲第10281号, Conferred date:2016/6/30, Degree Type:Course doctor, Examiner:,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

DEMO-based Information Systems Development

– Enterprise Ontology Oriented Approach –

In partial fulfillment of the Degree of Ph.D. of Engineering

By

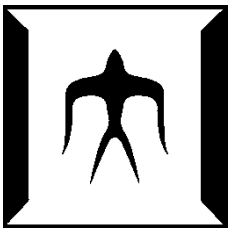
Tarek Fatyani

Supervised by Professor Junich Iijima

A dissertation submitted to the

Department of Industrial Engineering and Management

Graduate School of Decision Science and Technology



Tokyo Institute of Technology

Tokyo, April 2016

ABSTRACT

Information Systems Development (ISD) has encountered a variety of challenges in identifying complex requirements from multiple stakeholders. Therefore, users and their information environments have been a central issue for developing an abstract model of the early stages of information system development. To address this issue, modeling methodologies have considered modeling the enterprise as a social system. DEMO is a social system modeling methodology that addresses diverse social theories and multiple stakeholders' requirements. Additionally, the DEMO model has improved not only the quality of the requirements but also reduced the time for implementation during the early stages of ISD. Therefore, this research focuses on developing an information system based on the DEMO model by deriving the conceptual artifacts of an IS from the DEMO model. First, to understand the advantages and disadvantages of DEMO, I conducted a comparison between DEMO and I star (i*) framework. Because the later one is widely used in information system development. And it is similar to DEMO in the concept of modeling enterprise as a social system. They focus on modeling the people and the interaction between them. The comparison is to highlight the strong and the weak part of both modelings. Moreover, this research draws guidelines for improving both methodologies in modeling enterprise as a prior step in developing information system. As a result, the concept of modeling the interaction between DEMO and i* is different. DEMO is more formal in modeling the interaction rather than i*. Moreover, DEMO models both the structure and the behavior through its different diagrams. But i* does not capture the behavior. In contrast, i* allows modeling the non-functional requirements, too. Sometimes it is useful to analysis them during the first stages of requirements analysis. Secondly, based on the result of the comparison, a framework for developing information systems based on DEMO is proposed.

The framework utilizes all the diagrams of DEMO to construct the important artifacts in information systems. Those are the use cases diagrams, entity relationship diagram and other related artifacts. Third, to analysis, the transaction pattern of DEMO and know how to present it in an information system, a case study of project management is used to develop a database schema based on the complete transaction pattern. Fourth, to maximize the utilization of DEMO, a simulation method of DEMO is proposed for validating the DEMO models in the real world that enable debugging and testing the model. The methodology is based on mapping one to one from the DEMO model to Coloured Petri Net. The reason for choosing CPN is to use the richness of the Petri Net research results on, e.g., performance, deadlock analysis, animation, etc. Furthermore, CPN has a mathematical representation, which can initiate research on analyzing DEMO models mathematically. Finally, based on real world project, this study proposes that use of the DEMO model could lead to important discussions associated with actor roles, responsibility, and authority. These discussions are critical when defining the objectives and requirements of any information system to be developed.

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to my research supervisors Professor Junichi Iijima for his patient guidance, enthusiastic encouragement and useful critiques of this research work. I would also like to thank Dr. Jaehyun Park, for his advice and assistance in keeping my progress on schedule. My thanks are also extended to Dr. Zakaria Abdelrasol for his help in project management research.

Moreover, I would like to thank my wife for being always with me and giving me the support I need. Thanks to her very much.

I would also like to extend my thanks to my fellow lab mates in for the stimulating discussions, for the sleepless nights we were working together before deadlines, and for all the fun we have had in the last four years.

Finally, I wish to thank my parents for their support and encouragement throughout my study and all my life.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 Complexity of Requirements in Information Systems Development

As enterprises have grown and complexity, the complexity, and pervasiveness of their Information Systems (ISs) have grown exponentially (Mylopoulos, Yu, Giorgini, & Maiden, 2011). Also, continual advances in technology require continuous improvement in existing ISs. Due to this complexity, IS project failures are an ongoing issue because of the misalignment of business and IT. For example, the Standish Group reported that only 32% of IT projects are successes (Paul & Tan, 2015), while Wright and Capps (Wright & Capps, 2010) stated, "20 – 30% of all IS development projects are perceived as overwhelming failures, while 30% to 60% are partial failures." One cause of business/IT misalignment is that IT systems' functionality often does not meet expectations. For example, a research case study of a UK public sector organization found that an estimated 60% of completed projects had not met the original objectives (Paul & Tan, 2015). Another cause of business/IT misalignment is the need for major modifications to IT systems after acceptance testing, which may result from the lack of early validation for IS functional requirements (Paul & Tan, 2015). Furthermore, IS complexity may result in cost overruns that exceed estimates (Dietz, System Ontology and Its Role in Software Development, 2005). Additionally, the cost of enterprise IT system maintenance increases exponentially over time, sometimes becoming unmanageable regarding both complexity and resources (Wright & Capps, 2010).

Therefore, developing ISs successfully is extremely challenging due to their complexity. It is generally agreed that the only realistic way to manage this complexity and continue to provide IS benefits is to develop ISs using appropriate methods of abstraction (Dietz, System Ontology and Its Role in Software Development, 2005). This goal requires an abstract model that enables people to understand an enterprise in a simple and concise way. Abstract models

are used primarily as communication and collaboration mechanisms when groups need to solve problems and/or obtain joint and mutual understanding of an overall design (Gorschek, Tempero, & Angelis, 2014). The use of abstract models for IS design promises improved productivity, product quality, and shorter lead times because of increased abstraction levels and reduced gaps between problems and solutions. Abstract models are also useful for maintenance and technical software documentation (Mallens, Dietz, & Hommes, 2001), (Fitsilis, Gerogiannis, & Anthopoulos, 2014), (Fernández-Sáez, Generoa, Chaudronb, Caivanoc, & Ramosd, 2015) and (Garousi, et al., 2015). These gains could be achieved by modeling the enterprise at a stage prior to the early development stages of an IS.

Although many modeling methodologies currently exist, including UML, IDEF, BPMN and EPC, current modeling methodologies are not sufficient to meet the IS community's needs (Petre, 2014). UML's complexity is another recurrent theme, especially the complexity attendant upon large models, a criticism that echoes through the literature. However, despite their maturity, these modeling languages still lack comprehensive constructs for representing some core business concepts (Letsholo, Chioasca, & Zhao, 2013). Although such modeling methodologies are supposed to support understanding the business domain using a more abstract model that is completely independent of implementation, these methodologies consider both the original facts and the implementation decisions made during the design phases of IS development but do not differentiate between them, which increases the complexity of IS development efforts. A relatively recent approach to coping with the aforementioned problems was proposed by Pandey et al. (Malta, et al., 2011). They developed i*, a modeling methodology that can represent how well the intended system meets organizational goals, why the system is needed, what alternatives were considered, what the implications of the alternatives were for the various stakeholders, and how the stakeholders' interests and concerns might be addressed. Very recently, in (Fatyani, Iijima, &

Park, Comparing DEMO with i_Star In Identifying Software Functional Requirement, 2015), a comparison between i* and Design & Engineering Methodology for Organizations (DEMO) showed that both are social modeling methodologies: they focus on humans and human interactions in their modeling. DEMO is implementation independent; therefore, the DEMO model does not need to change before or after implementing any IT solutions. However, i* is an implementation dependent methodology. DEMO provides a more formal and rigorous model of the enterprise, which makes it a good potential modeling methodology for understanding an enterprise before implementing any IT solution.

1.2 Utilizing DEMO in Developing Information Systems

DEMO is built on the Performance in Social Interaction (PSI) theory and provides a consistent, coherent, concise, comprehensive and essential (C4E) representation of the system. The DEMO model is both abstract and implementation independent, and it tends to provide a mutual understanding of the business. DEMO has already proved its usefulness in enterprise engineering, yet developing ISs based on DEMO is still challenging. These problems occur precisely because DEMO is an implementation independent model. DEMO does not consider methods of implementation; instead, it focuses on the ontological level of the enterprise. Therefore, implementation-related design decisions are deferred and must be made during IS development. There are several studies related to using DEMO for IS development available in the literature, but these studies discussed only the theoretical side and provided no guidelines or practical advice on developing IS-suitable DEMO models or transferring existing models to an IS.

1.3 DEMO Models Simulation

Enterprises are growing in complexity due to the existence of many business processes that form an interwoven network of business transactions. To design and re-engineer such an

enterprise, a conceptual model of the enterprise is needed. In recent decades, many modeling methodologies have been developed. Among those methodologies is DEMO. Design and Engineering Methodology for Organizations (DEMO) is a methodology for the design and engineering of organizations. DEMO is a concise and coherent model that illustrates the essence of an organization (Dietz, Enterprise Ontology Theory and Methodology, 2006). However, DEMO lacks tools that support the simulation of its models. DEMO Simulation provides a powerful tool to validate the proposed DEMO model compared to the real world by running, debugging and analyzing the model. Deadlocks or any unpredicted roots can be discovered during the simulation. Furthermore, simulating DEMO models may answer "what if" questions, which may be a useful tool for re-engineering the enterprise. Additionally, Petri net is a simple modeling language used to model and analyze concurrent systems. Its simplicity and simulation capability make it appealing for many other modeling languages. They transfer their model into PN to utilize its features, e.g., Activity Diagram and BPMN. However, the usefulness of the simulation model depends on the quality of the original conceptual model like AD or BPMN. And if those models don't represent the ontology of the enterprise, then the quality of the simulation models will be low. Therefore, an conceptual ontology model is needed as a base to do the simulation. We call it here ontology based simulation. By analyzing PN and DEMO models, the similarity between the two is clear. The concepts of Fact and Act in DEMO model can be perfectly mapped to the concepts of Place and Transition in PN. This similarity creates the potential to map DEMO to PN, which is useful not only for simulating DEMO models but also for utilizing the richness of research and analysis that are conducted on PN, as well as other tools that have been developed for those purposes. Therefore, in this research, a transformation methodology from DEMO to Coloured Petri Net is introduced. The transformation mainly focuses on the Process Model (PM) of DEMO, because PM describes the dynamics of the enterprise as a workflow and is

used to create the Basic Petri Net. However, the State Model (SM), and the Action Model (AM) of DEMO are also used to define the Color Sets, Guards, Variables and Expressions in CPN. The potential benefits from transforming DEMO into CPN can be summarized as follows: first, CPN is based on Petri net, which can be used for simulation. Therefore, the transferred model can be used to simulate the DEMO models. Second, CPN is an old and simple process modeling language that provides a lot of expertise and tools that can be used for analysis of its models. Third, CPN has a mathematical representation that can lead to interesting research that analyses DEMO models mathematically. Reasons for choosing Coloured Petri Net include its ability to capture the cardinality in DEMO and its ability to program the business rules in the AM of DEMO by constructing the Guards and Expressions in CPN. Therefore, the main contribution of this research is the transformation of the DEMO model into CPN and showing the validity of our transformational approach by implementing one case study.

1.4 Overview of the Dissertation

This research provides a practical framework to illustrate the stages involved in using the DEMO model and for developing ISs based on it. Furthermore, this dissertation provides practical guidelines for developing DEMO models that function as good abstract models for developing ISs. The remaining chapters of the research are organized in the following way: chapter 2 provides a review of the current literature concerning process modeling approaches that focus on DEMO. Additionally, it provides a critical review of the most closely related work. A brief overview of DEMO, the proposed framework, and some guidelines are presented in chapter 3. Chapter 4 compares using DEMO models in ISD to using i* framework in ISD to highlight the strengths and weakness of DEMO in ISD. Chapter 5 provide a framework for simulating DEMO models by transferring them to colored Petri net

models as a pre-stage of ISD. This step is important when the model is complex and need validation before starting the development of IS. Then, a real world case study is introduced in chapter 6. Related discussions, results, conclusions, and future work are covered in chapter 7.

CHAPTER 2: LITERATURE REVIEW

2.1 DEMO Methodology and Enterprise Ontology

Enterprise ontology is a core theory of enterprise engineering. In particular, the goal is to provide a new understanding for enterprises, penetrating the distracting and confusing appearance of an enterprise and revealing its underlying essence. Enterprise ontology DEMO models have based on the PSI (Performance in Social Interaction) theory. In PSI theory, an enterprise (organization) is considered to be an interaction of individual social subjects (Dietz, System Ontology and Its Role in Software Development, 2005). DEMO helps in "discovering" an enterprise's ontological model primarily by re-engineering it from its implementation.

In DEMO, an organization is composed of actors (human beings) who perform two types of acts: production acts (P-acts), through which the actors contribute to bringing about the goods or services that are delivered to the environment, and coordination acts (C-acts), in which actors enter into and comply with commitments, initiating and coordinating the execution of production acts. C-acts and P-acts occur in universal patterns called transactions. Figure 2-1 presents the basic transaction pattern.

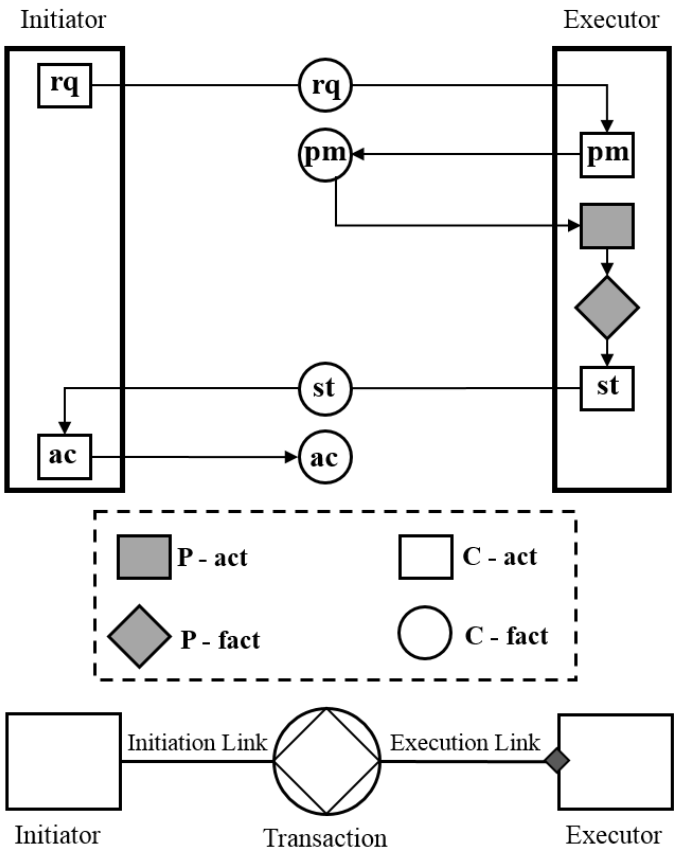


Figure 2-1 Basic transaction pattern

A transaction involves two subjects: an initiator, who generates a request, asks for a production and commits to the delivered result, and an executor, who produces products, gives feedback to the request and is responsible for producing the result (Dietz, System Ontology and Its Role in Software Development, 2005). Transactions are the elementary (essential) organizational building blocks of enterprises. There are three patterns of transactions: basic, standard and complete. The basic pattern includes only the happy path while the standard pattern covers negative acts such as “refuse” and “decline.” The complete pattern includes both completion and revocation acts, allowing actor roles to revoke their acts. DEMO differentiates between three transaction levels: ontological, infological and datalogical. An ontological transaction is one that creates the original artifact as a result of executing it. An infological transaction involves a derived artifact such as a calculation. A

datalogical transaction is a transaction that merely involves storing and retrieving data without changing it. In DEMO, ontological transactions are denoted by the color red, infological transactions by green, and datalogical transactions by blue.

To obtain a full image, the ontological model of an organization is divided into four sub-models describing different aspects of the complete model, Figure 2-2.

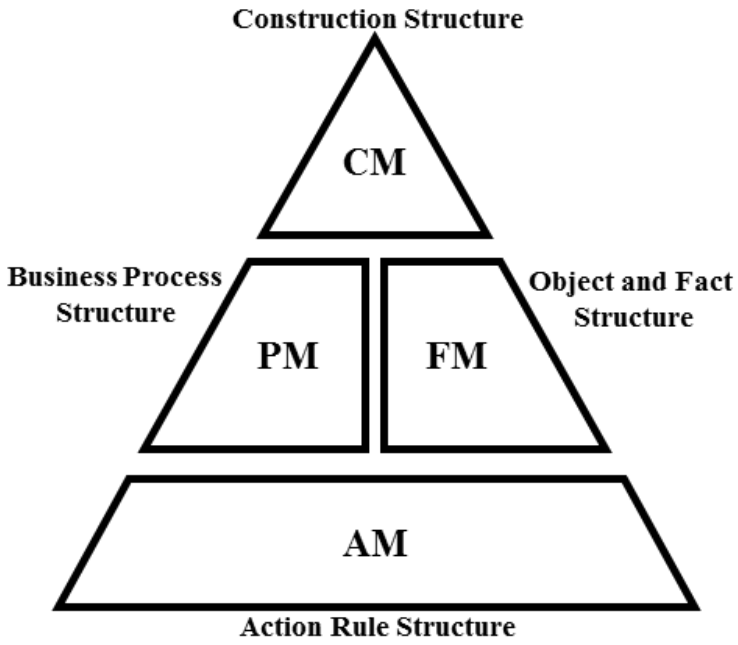


Figure 2-2 DEMO aspect models

The Construction Model (CM), located on the top of the triangle, is the most concise model. The CM provides a general view of the enterprise by showing the transactions, the actor roles, and how they are interrelated with and composed to construct a system. The Process Model (PM) provides more details about the business processes and business events. The PM describes the detailed causal relations between the transactions. The Fact Model (FM) illustrates the object classes with their related properties and facts in the process. Finally, the Action Model (AM), located at the base of the triangle, describes the action or business rules for actor roles. These business rules govern the actions between the process steps. With the four DEMO aspect models, DEMO provides a consistent, coherent, concise, comprehensive

and essential (C4E) representation of the system. The details of the DEMO aspect models are explained by constructing and analyzing DEMO models for the PMS process.

2.2 Developing Information Systems based on DEMO

Developing ISs is a challenging task because of their complexity. It is generally agreed that the only realistic way to manage this complexity and to continue to provide IS benefits is to develop ISs using appropriate methods of abstraction (DE JONG, A Method for Enterprise Ontology based Design of Enterprise Information Systems, 2013). The literature is rich with a variety of modeling methodologies that support the many different modeling needs in IS development. First, we present and then briefly discuss the literature. Second, we provide a critical review of the papers most closely related to our work.

Business process modeling is essential in IS development. During the early stages of the IS development cycle, a domain model is a type of artifact that captures the common and variable aspects of software products (Reinhartz-Berger & Sturm, 2012). A study conducted by Shishkov & Dietz, 2001 (Shishkov & Dietz , Analysis of Suitability, Appropriateness and Adequacy of Use Cases Combined with Activity Diagram for Business Systems Modeling, 2001) showed that use cases represent a promising and essential tool for business process modeling because they can help with visualizing models of the business processes under study. The study also proposed that use cases could be combined with activity diagrams for building more consistent and complete models of a system and representing different system-actor interactions. However, use cases are intended to capture the functional requirements of ISs; identifying use cases at the model stage is not so simple. To tackle this issue, Dietz, 2003 (Dietz, Deriving Use Cases from Business Process Models, 2003) proposed an approach that derives use cases from business system models produced by applying Demo Engineering Methodology for Organizations (DEMO). To derive the use cases from the models, they proposed a three-step procedure by which the derived use cases retain the same properties of

the essence, atomicity, and completeness found in the models. Furthermore, recent research by Bera & Evermann, 2014 (Bera & Evermann, 2014) specified the semantics of various UML language elements for domain modeling, based on ontological considerations. They empirically examined ontological modeling guidelines for the UML association construct, which plays a central role in UML class diagrams. Using an experimental study, they found that some of the proposed guidelines led to better application domain models. They also found that ontological guidelines could improve the usefulness of UML class diagrams for describing the application domain, and thus have the potential to improve downstream system development activities and ultimately affect the success of an IS implementation.

A relatively recent approach, i*, has been applied by many authors to cope with the aforementioned IS development problems. Alencar et al. (Alencar, Silva, Moreira, Araújo, & Castro, 2006) used the i* models to identify crosscutting concerns early in the software development process. To accomplish this, they started by analyzing the strategic dependencies and strategic rational models to identify those model elements that might cut across several other elements. The result was validated by applying a set of heuristics to derive a use case model from the i* models and then identifying potential crosscutting concerns in the resulting UML model. For business/IT alignment, Maz´on et al. (Mazón, Pardillo, & Trujillo, 2007) used the i* modeling framework and the Model Driven Architecture (MDA) approach to describe (i) how to model goals and information requirements for data warehouses, and (ii) how to derive a conceptual multidimensional model that provides the information required to support the decision making process. Furthermore, António et al. (António, Araújo, & Silva, 2009) aimed to benefit Software Product Lines (SPLs) from the i* framework, taking a more expressive approach toward requirements engineering for SPLs. Thus, they extended the i* framework by adding

cardinality to the intentional elements, making it easier to derive a feature model from the i* models.

Many authors have applied DEMO to support IS development. One frequent cause of software project failure is the mismatch between the (business) requirements and the actual functionality of the delivered (software) application. Shishkov & Dietz, 2004 (Shishkov & Dietz, Design of Software Applications Using Generic Business Components, 2004) proposed an approach to software design that is consistently based on prior business process modeling. The alignment between these two tasks is realized in a component-based manner, by deriving the software model from identified (generic) business components, thus taking advantage of the benefits of object-orientation. However, their study was both theoretical and abstract rather than practical; they expected that the proposed approach would be a useful contribution to the knowledge on aligning business process modeling and software design. In another study (Shishkov & Dietz, Deriving Use Cases from Business Processes, 2004), authors aimed to discover how to find all relevant use cases, based on sound business process modeling. They studied and analyzed DEMO's strengths concerning the derivation of use cases. Shishkov & Dietz, 2005 (Shishkov & Dietz, Applying Component-Based UML-Driven Conceptual Modeling in SDBC, 2005) extended the SDBC approach by creating step-by-step methodological application guidelines on how to accomplish this. However, their study is also theoretical and abstract rather than practical.

Starting from the notion that the direct mapping of business processes to business components often leads to inadequate results, a methodology for identifying and refining business components based on the functional decomposition of an application domain was proposed by Albani et al. (Albani, Keiblinger, Turowski, & Winnewisser, 2003). To support collaboration between business partners, adequate ISs must be built to automate inter-organizational business processes. To guarantee the appropriateness and quality of the

underlying business domain models, Albani et al. (Albani & Dietz, Identifying Business Components on the Basis of an Enterprise Ontology, 2005) introduced a process for identifying business components based on enterprise ontology, assuming a business domain model that satisfied well-defined quality criteria. However, their study is also theoretical and abstract.

To avoid a mismatch between the provider's intent and the consumer's expectations concerning the functionality of a corresponding service, Terlouw and Albani (Terlouw & Albani, 2013) proposed an enterprise ontology-based approach for service definition, service classification, and service specification. They concluded by deriving the information required for each service from the transactions.

To respond effectively to environmental changes, such as changes in market needs, technology, regulations or law, enterprises need to be able to modify their supporting information system(s) accordingly. Krouwel and Land (Krouwel & Op 't Land, 2011) aimed to find key concepts to link agile enterprises with agile automated ISs by combining DEMO and Normalized Systems. They found that DEMO and its underlying PSI-theory matched the principles and elements of the Normalized Systems approach. Also, using two cases of Dutch governmental subsidy schemes, they designed a few automatable steps to derive a Normalized System from the ontological model of the B-organization, provided by applying DEMO to an enterprise, while retaining the implementation freedom of the organization under consideration. However, their approach is too complex, and there are no guidelines.

In the available literature, three publications are closely related to this study. We discuss these critically below.

Albani and Dietz (Albani & Dietz, Enterprise Ontology Based Development of Information Systems, 2011) demonstrated an Information System Development (ISD) methodology based

on the notions of enterprise ontology and business components and explained it within a conceptual framework called the generic system development process. The methodology allowed both for a reduction in the complexity of domain models and for identification of stable business components. Furthermore, the resulting IS models contain only their essential features, which appear to be quite understandable by business people. However, there are some shortages in their methodology. First, they presented only general concepts about function and construction perspectives. Second, it is too abstract; concrete details about actual performance are lacking. Third, the relation between green and red is not simple; instead, it is a network.

Guerreiro et al. (Guerreiro, Kervel, & Babkin, 2013) conceptualized an architectural framework specifically designed for explaining and representing the working environment for enterprise operating systems (EOS) and non-EOS ISs that depend on them. They proposed a new conceptual structure for software architectural frameworks. Such frameworks are intended to reduce complexity and ambiguity during software engineering for several classes of IS. Their proposed framework is a good simulation tool, is easy to model, and its DEMO processor is easy to implement; however, there are some omissions. First, the framework did not consider the rational aspects of ISs. Second, it focused only on the CM and the PM and did not use the FM. Third, it captured only the coordination facts—not the production facts. Finally, it must be linked with other systems.

Jong (DE JONG, Designing the Information Organization from Ontological Perspective, 2011) proposed the PID framework, which guides the design of enterprise information systems based on enterprise ontology. He provided general guidelines for implementations of PID and also proposed integrating the three layers of DEMO models—the business, information and data layers. While the relationships between the three layers are interesting, from a practical point of view only the business layer (with a few information layer

transactions) should be modeled. Including the entire set of possible transactions and including the data layer only makes the model more complex. Moreover, the original problem involves making the model simpler and focusing on only the important aspects of the enterprise. Another point missing from this research is the linkage between the DEMO models and other IS-related models. This linkage makes it very difficult for IS engineers to adopt the system utilizing only the DEMO models.

The following three points sum up the current gaps in the literature related to developing ISs based on DEMO. First, an IS is a mixture of rational systems executed by computers and social systems composed of people interacting with the system. There is a need to capture both the social and rational aspects in the model. Second, most of the research focuses on the theoretical aspects, creating a gap between research and industry. There is a need to justify these theoretical frameworks through real-life cases studies. Third, there is a need to utilize all the aspects of DEMO models in developing an IS, including not only the functional aspects of the IS but also how to develop the database schema from the Fact Model in DEMO.

To fill these gaps, this research proposes a practical framework for developing ISs based on DEMO. A real world case study follows that shows how to utilize the framework.

CHAPTER 3: DEMO BASED ISD METHODOLOGY

3.1 The General Framework for ISD based on DEMO

This chapter introduces our framework for developing ISs based on DEMO models. The purpose of this framework is to reduce the complexity of the proposed IS during the first stages of the analysis and design phases. This reduction of complexity is not only important for developing good software but also has a crucial impact on the future maintenance of the developed software (Williams & Carver, 2014). Our framework is based on DEMO, which has proved its usefulness in reducing the complexity of the enterprise in both process reengineering and in software development (Dumay, Dietz, & Mulder, 2005), (Albani & Dietz, The Benefit of Enterprise Ontology in Identifying Business Components, 2006) and (Op ’t Land & Dietz, 2012). The framework derives IS artifacts from DEMO models. Those artifacts are composed of UML because UML is already familiar to IS developers, and the use of UML diagrams has a good impact on source-code comprehensibility and modifiability (Scanniello, Gravino, Genero, Cruz-Lemus, & Tortora, 2014). Next, we provide workflow guidelines to illustrate a practical method of implementation. We assume that the DEMO model has already been completed in a previous stage. Therefore, this framework does not deal with constructing the DEMO model; instead, it focuses on utilizing the DEMO model in developing an information system.

Figure 3-1 presents the whole picture of the framework. I distinguish between too levels, the enterprise business and the information systems. Enterprise business focuses on the business not the implemented information systems. First we model the enterprise business by using DEMO, and then developing the information system can be processed.

The traditional way of developing information system is to get the requirements from business and then develop the information system. But in this research, I model the enterprise before starting designing the information system. This is a very important step to make sure that a comprehensive model of the enterprise is understood by all the stockholders. To validate the DEMO models, a simulation methodology is proposed in this research in chapter 5. And to highlight the importance of DEMO in information system development, a comparison between DEMO and i* is presented in chapter 4. It explains the reason why to choose DEMO instead of i*.

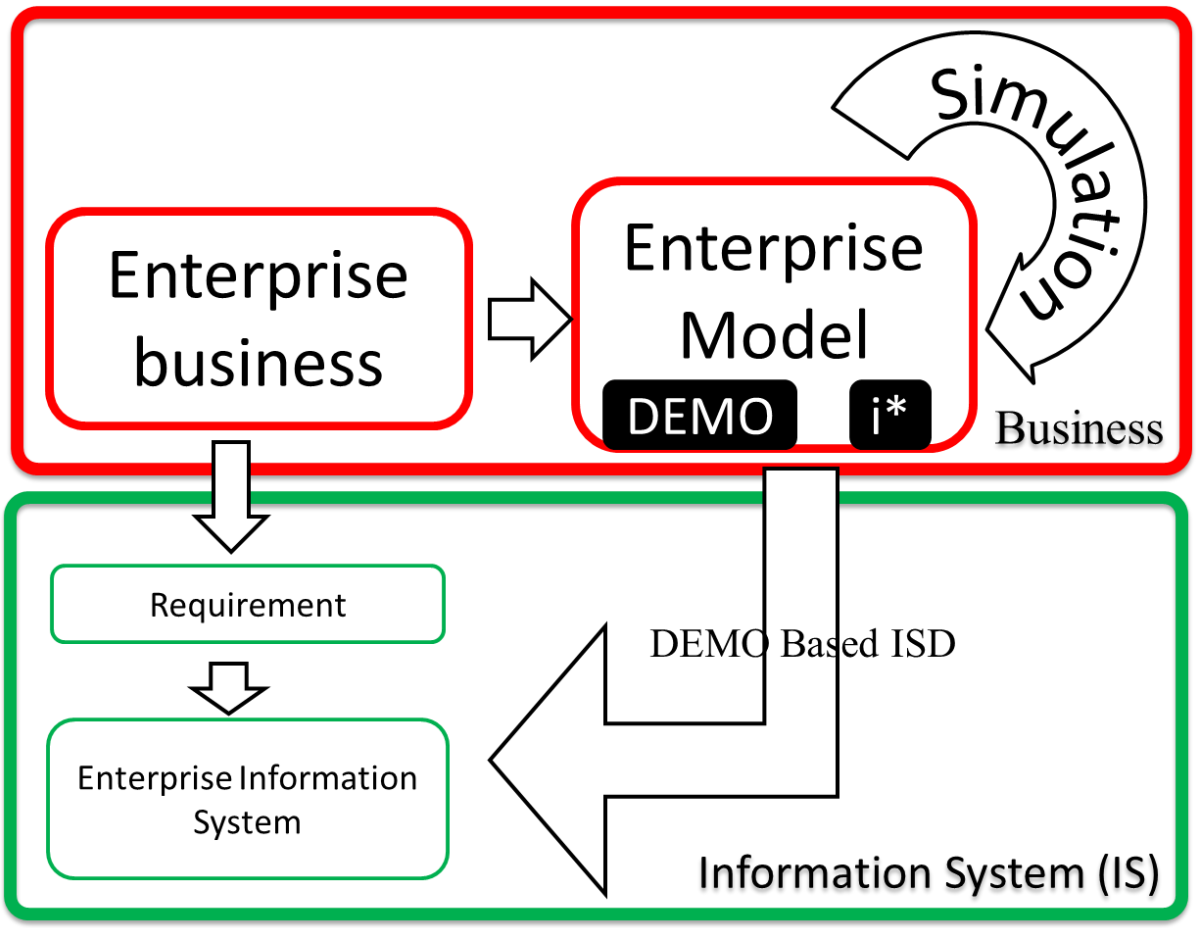


Figure 3-1 The general framework for ISD based on DEMO

3.2 The Relationships between the Artifacts from DEMO and IS

Figure 3-2 shows the relationship between the DEMO artifacts and those of the information systems. The DEMO artifacts are its four aspect diagrams: the Construction Model (CM), Fact Model (FM), Process Model (PM) and Action Model (AM). The artifacts for the IS used here are use case diagrams and scenarios, activity diagrams and entity relationship diagrams. The CM contributes directly to the derivation of use cases. For each transaction, there will be a group of use cases that meet the functionality of this transaction. The PM contributes only partially to the derivation of use cases. This is because not all the process steps in the transaction need to have a use case; whether a use case is needed depends on the requirements that specify how the users will interact with the IS.

Both the PM and the AM contribute toward drawing needed activity diagrams. There is no need to draw activity diagrams for all the cases. We draw them only where it is necessary to illustrate a particularly important series of activities.

The AM presents the pre and post conditions for each act. Therefore, it contributes to the use case scenarios. The FM represents the objects that are used in the system; therefore, ERDs are derived directly from it.

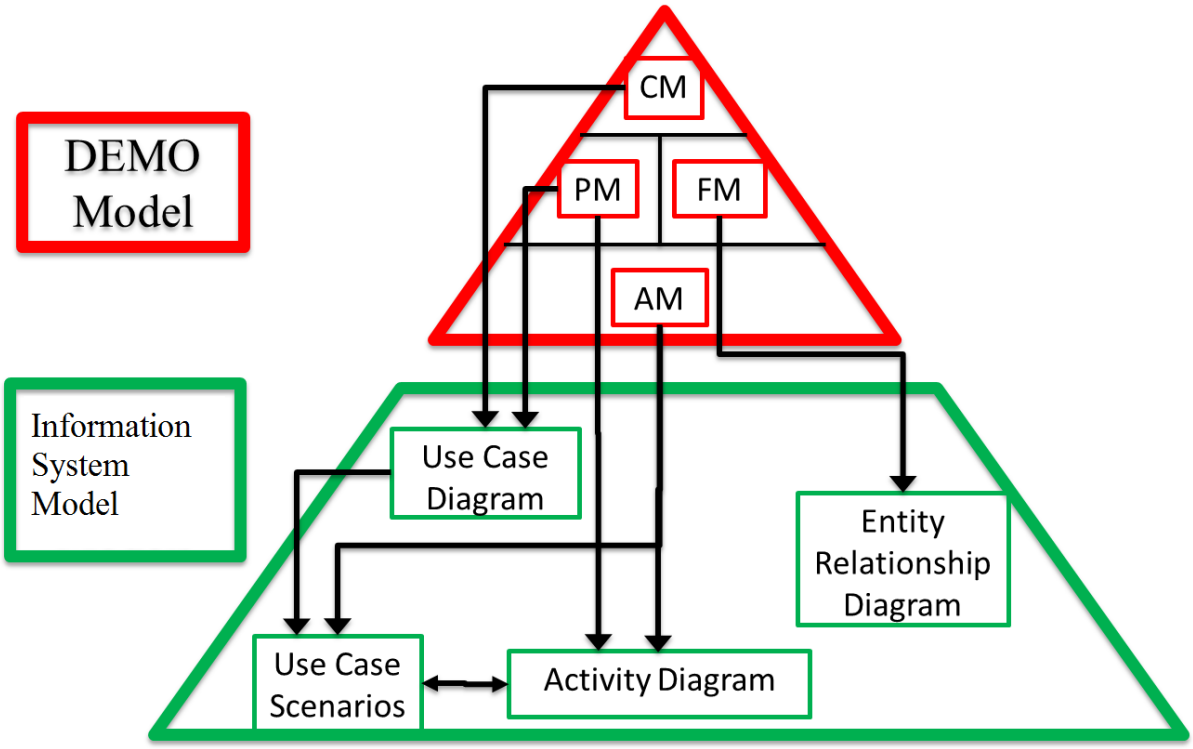


Figure 3-2 the relationships between the artifacts from DEMO and IS

3.3 Steps for ISD based on DEMO

Figure 3-3 illustrates our approach by presenting the various stages involved, starting by investigating the scope of the development project (Fatyani, Iijima, & Park, Enterprise Ontology-based Information Systems Development, 2015). At this stage, the questions to answer are: "What is the scope of the system?" and "What are the boundaries of the system?"

The second stage involves analyzing the users of the system. Who are the actors in this system? What are their roles in the system? These questions will be answered by analyzing the actor roles of DEMO and matching them with the actors of the enterprise. This analysis is called Actor Role-Actor analysis. Any delegation identified should be specified at this stage. Transaction patterns could be used as a reference to identify the possible process steps and who will act on them.

Third, the functional requirements are specified by deriving the use cases from the transactions. The details of each use case are specified by its scenario. We may draw an activity diagram for difficult use cases.

Fourth, the data structure is developed. The Product Structure shows the entire image of the structure of the system and represents the relationships between the transactions. Then, an Entity Relationship Diagram (ERD) will be developed based on the fact model. The ERD is then expanded by defining the entities and their attributes through entity annotations.

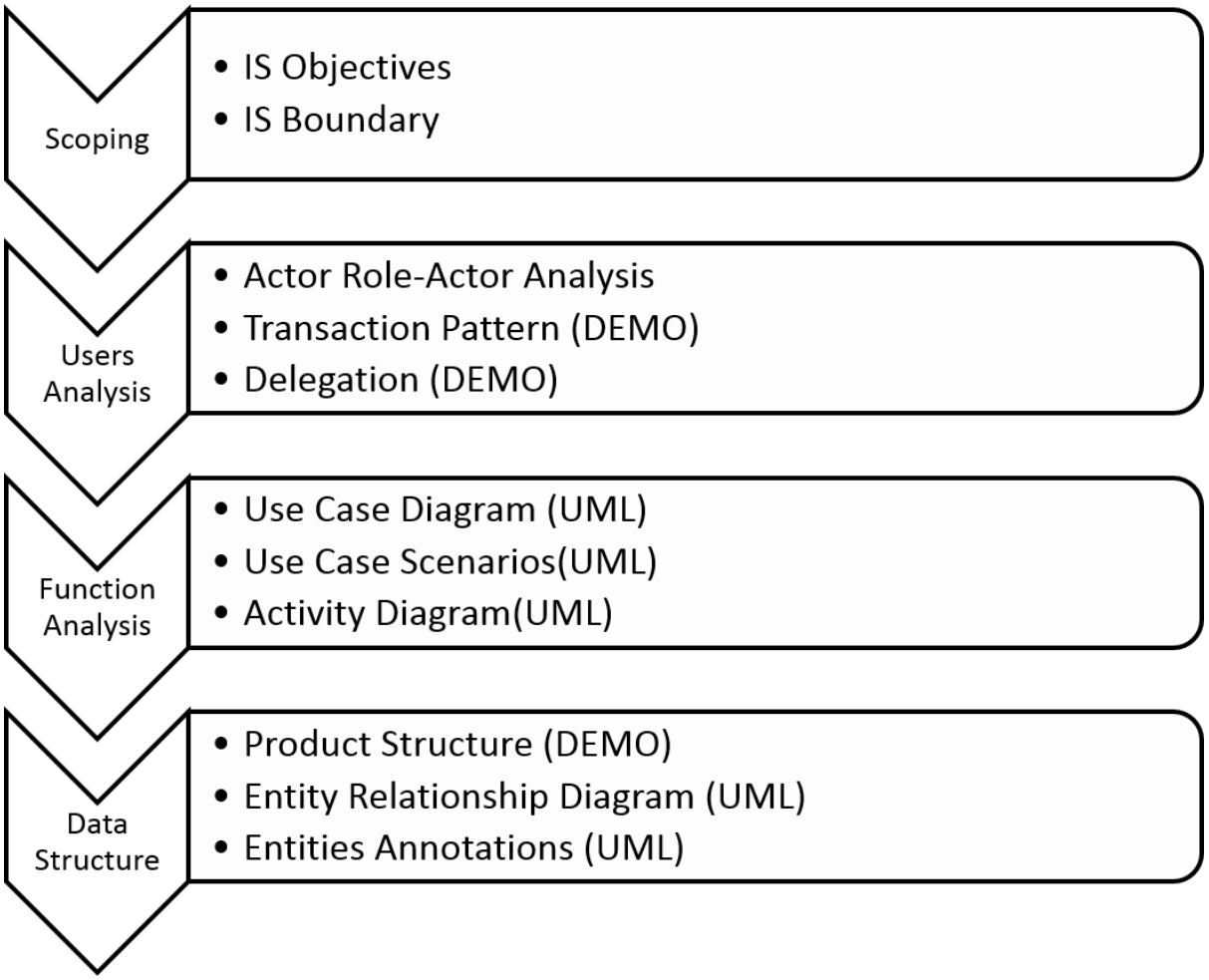


Figure 3-3 Steps for ISD based on DEMO

The steps are explained in detail as follows:

STEP 1: SCOPING

IS Objectives: Setting the objectives of developing the information system must be completed first of all. We must know what the software is expected to do. Why do we need this? It is a common mistake to think that the objective is to automate every process in the enterprise that can be automated. Instead, we must set the objectives based on needs, budgets and time and other resource limitations. One important point in this step is to make sure that the DEMO model covers all the transactions that are related to the objectives.

IS Boundaries: Boundaries are defined as the limits of the system, where the system meets the environment. Defining the boundary is important in ISD so that developers know what functions they must include and what not to include. Using the DEMO model, it is easy to define the boundary; we simply need to specify which transactions are included in our system and which transactions are out of scope. The boundary should be precisely defined based on the objectives set in the previous step.

STEP 2: USER ANALYSIS

Actor Role-Actor analysis: DEMO provides the elementary actor roles. However, during the implementation, real users fulfil these actor roles based on the business context. An analysis of who will fulfil each actor role is important during implementation. This could be accomplished by creating an actor role-actor table in which each row contains one actor and the actor roles that he or she fulfils. Generally speaking, one actor fulfils one or more different actor roles; however, sometimes one actor role might be fulfilled by two actors. This must be defined clearly. After implementation, during usage, the actor roles that were assigned to one actor might change. When this occurs, a control panel must be developed to allow redefining the actor roles assigned to each actor. This analysis is critical to define the authentications and permissions in the information system.

Transaction patterns: DEMO methodology provides three patterns of the transaction, named basic, standard and complete. The basic transaction pattern includes only the process steps that are in the happy path. Declining an initiator's request or an initiator rejecting the result of a transaction are not included in the basic pattern. The basic pattern is useful when the execution path is clear, and there is no possibility for disagreement between the initiator and the executor. When a possibility for declining the request or rejecting the result of the transaction exists, the standard pattern should be used instead of the basic one. Using the standard pattern, a transaction has three possible outcomes: quit, accept or stop. If revoke capability is required in the IS for a partial transaction, then the complete transaction pattern should be used. This pattern is the most comprehensive and contains all the possible scenarios. Determining which pattern to use for each transaction is based on the business needs analysis. Note that while using the complete transaction pattern for all the transactions might seem like a good idea, doing so makes the IS very complex, and it will contain unneeded scenarios. Therefore, an analysis of which pattern to use must be performed for each transaction.

Delegation: During user actor role analysis, every user will be matched to the actor roles that he or she will fulfil. However, sometimes an actor may delegate his or her actor role to another person. Such delegation might occur for all the process steps of that actor role or it might occur for only one process step, such as a request. Any delegation should be defined clearly in this stage so that the information system implementation will support the delegations.

STEP 3: FUNCTION ANALYSIS

Use case diagrams: Use case diagrams (UCDs) are popular in ISD because they define the functionality that the information system must provide to the users. A UCD is a graphically

based description of the system functions, and it uses natural language. Therefore, it is easy to understand and easy for both domain experts and design experts to validate.

Use cases are not only useful for implementation but are also used for testing the IS after development and for documentation purposes. UCs can be derived from the CM and PM models. When developing UCs from the DEMO model, one should bear in mind that this process is not a one-to-one mapping. It needs analysis and decisions by domain experts and design experts. However, having the DEMO model makes this process easier. Each red transaction needs one or more than one use case. Generally speaking, every red transaction needs one use case to initiate a new artifact, edit an existing artifact, delete an existing artifact, view the details of an existing artifact and view the list of existing artifacts. Depending on the transaction, some may require other use cases, such as calculating derived artifacts. Green transactions do not create a new artifact; therefore, only viewing or calculating use cases can be derived from green transactions. The derivation process is shown in Table 3-1. Note that additional use cases are always needed in every IS. The use cases listed in Table 3-1 are user- and system-related use cases.

Table 3-1: Deriving Use Cases from DEMO Transaction

DEMO Transactions	IS Use Case
Business (ontological) Transaction (red)	Common Use Cases: 1. Initiating a new transaction 2. Editing an existing transaction 3. Deleting an existing transaction 4. Viewing the details of an existing transaction 5. Viewing a list of existing transactions Specific Use case: Based on special user requirements.
Infological Transaction (green)	One use case with the same name.
	User-related UCs: 1. Add User 2. View user details 3. Edit user details 4. Delete User 5. View Users list 6. Add role to user 7. Edit user role System UCs: 1. Change password 2. Sign in 3. Sign out

Use case scenarios: A use case scenario defines the details of the use case. It defines the pre and post-conditions as well as all the possible steps that may occur during execution of the use case. All these details can be derived directly from the AM in DEMO. Not every process step in the DEMO PM is transferred to a use case, only the process-related step action models.

Activity diagram: Activity diagrams show the flow of activities in a use case scenario graphically. An activity diagram is used when the use case scenario is complex and needs to

be carefully validated. When an activity diagram is needed, it can easily be derived from the use case scenario and the AM.

STEP 4: DATA STRUCTURE

Product Structure: A product structure shows the relationship between the products of each transaction. It is a tree-based structure that helps in understanding what the root transaction is that triggers others. A product structure is based on the composite axiom. It represents the child-parent relationships between the products of the transactions. Product structures are important for building the entity-relationship diagram as well in dividing the system into components.

Entity Relationship Diagram: An entity relationship diagram defines the entities that are handled by the IS and illustrates how these are structured by defining the relationships between entities. A database system is at the base of an information system. Relational databases are widely used in various areas. The Entity-Relationship Diagram (ERD) is a common technique used to define data structures and for database systems design (Entity-Relationship Diagram, 2009). A basic ERD can be easily derived from the FM in DEMO by following the guidelines shown in Table 3-2. Every object in the FM except time-related objects will become an entity in the ERD. Time-related objects become attributes of other entities. The FM contains the passive entities of the system; therefore, to define the active entities, one can derive them from the user actor role analysis as shown in Table 3-2. All the properties of an object in FM will be attributes of the same entity in the ERD.

Table 3-2: Deriving Entities from DEMO Objects

Objects in DEMO	Entities in ERD
Red Object (not time related)	Entity (Extra entities may be needed based on user requirements)
Red Object (time related)	Entity Attribute
Green Object	_
	Users
	Roles
	Roles/Users

Entity annotations: To make sure that both the domain experts and design experts share the same understanding of the system to be developed, we strongly recommend building a glossary of entity annotations for the terminology used in the ERD. This could be created easily by providing a definition for each entity and each attribute.

CHAPTER 4: COMPARING DEMO WITH i* (I STAR) IN ISD

Current information systems are getting more complex in terms of the number of the stakeholders who benefits from these systems as well as in terms of the related information to be in the system. Therefore, information systems development (ISD) encountered a variety of challenges in terms of identifying the requirements among multiple stakeholders. How can one differentiate between what the users want and what they really need? One of the common problems in requirement analysis is requirements conflicts. Therefore analyzing the requirements is crucial for software development (Mazón, Pardillo, & Trujillo, 2007). Scoping and requirements engineering are the most important challenges that SMEs faces during information systems development (Silva, Neto, O'Leary, Almeida, & Meira, 2014). Therefore, before the requirement analysis stage, an abstract model of the enterprise is needed. This model must describe the essence of the enterprise. It should describe the structure of the organization and its interaction with its environment (Tuunanen, Rossi, Saarinen, & and Mathiassen, 2007). Failure to do so may lead to a requirement uncertainty (Michalik, Keutel, & Mellis, 2014).

In the last two decades, practitioners and researchers seek an alternative for modeling the enterprise as a social system. This means that enterprise consists of individuals who interact with each other to deliver a particular product or service to the environment. Therefore, new modeling methodologies were developed to model the enterprise as a prior model to any implementation. It has proved that such a modeling with analysis provides benefits at every stage of the requirements engineering process (Mazón, Pardillo, & Trujillo, 2007). DEMO and i* are good examples of these methodologies. DEMO provides a formal model of the enterprise, including the structure and the behavior. Although DEMO is based on strong theories, it is yet to be used in many real-world scenarios in developing information systems

(Kervel, John , Meeuwen, Vermolen, & Zijlstra, 2011). On the other hand, i* is widely accepted modeling methodology in many fields (Mylopoulos, Yu, Giorgini, & Maiden, 2011). In particular, it used in modeling the goals of the information system before developing it. There are many frameworks for developing the requirements based on i*. It is similar to DEMO in providing a better understanding of the decision-making process and the rationales behind it by providing an abstract model of the enterprise (Malta, et al., 2011).

This research aims to understand the difference in popularity of the two methodologies i* and DEMO by comparing them in one real case study. By this comparison, we can highlight the pros and cons of using DEMO. It also helps in developing a framework for developing information system based on DEMO similar to the frameworks that i* has.

As a result of this research, it is clear that DEMO is implementation independent methodology. But i* is implementation dependent methodology. This means that DEMO model does not change according to the implementation method. It is up to the designer to select the implementation that fits the enterprise needs. And DEMO model will not be changed before and after the implementation, unlike the i* model. However, i* can capture not only the social aspects of the enterprise but also the rational aspects, too. The rational dependency model of i* can model the processes as they are in the implementation. This is useful for developing the information systems.

The rest of the paper is as follows. First, literature review provides an explanation about i* and DEMO with their recent research in the field of requirements engineering. Second, a real world case study is introduced, then modeled by both i* and DEMO. The third is the conclusion with the discussion about the similarities and the differences between i* and DEMO.

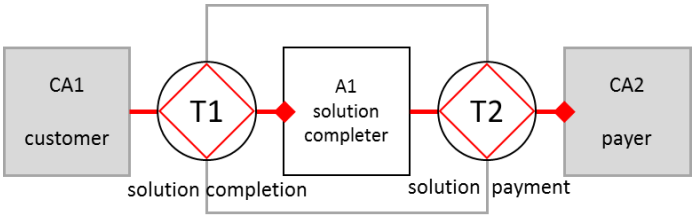
4.1 Background of the Two Modeling Methodology

In this section, the main concepts of DEMO and i* are explained. a running example will be used to explain the way of modeling of DEMO and i*. Where customers request the enterprise a particular IT solution. And they pay the fees for this service.

DEMO MODEL

DEMO, which stands for "Design and Engineering Methodology for Organizations", is based on PSI (Performance in Social Interaction)-theory. In this theory, an enterprise (organization) is considered as an interaction of individual social subjects. DEMO helps in 'discovering' an enterprise's ontological model, basically by re-engineering from its implementation. The main elements of DEMO models are actor roles and transactions. Any transaction within an enterprise is carried out by an interaction of two actor roles. The first actor role is responsible for initiating the transaction while the other actor role is responsible for executing the transaction (Dietz, Enterprise Ontology Theory and Methodology, 2006).

DEMO consists of four models. The construction model (CM) specifies the structure of the enterprise in relation to its environment, including the transactions, actor roles, information banks and links between them. The process model (PM) specifies the details of the transactions in the CM. Though the CM does not specify a sequence in which the transactions are executed, the PM does while indirectly indicating the timeline. The fact model (FM) specifies the object classes, which consist of a fact kinds and transaction result kinds. The action model (AM) formulates the business rules for executing each process step in the PM.



transaction kind	product kind
T1 solution completion T2 solution payment	P1 Solution **is** completed P2 **the** fee **of** Solution **is** paid

Figure 4-1: CM model of simple case

Figure 4-1 shows CM model of a simple case. CM consists of OCD (organization structure diagram) and TPT (transaction product table). Customer requests a solution from IT Department store, which is represented, by the actor role solution completer. This actor role is the executor of T1. Therefore, a small diamond appears at the end of the link to T1. The solution completer then asks for the payment by initiating the transaction T2. The gray rectangle represents the scope of interest (IT Department store). "A" stands for actor role. White actors are elementary actors. And gray actors are composite actor roles. TPT shows the product of each transaction after completion.

In DEMO, the ontology, infology, and datalogy levels are clearly differentiated. In the ontology level, actors (human beings) initiate and execute transactions that result in original facts, for example, purchasing. At the infology level, transactions only manipulate information from one shape to another, for example, calculating salary. At the third level, datalogy, transactions store and retrieve data without any manipulation. DEMO provides a high level of abstraction of the enterprise.

DEMO has already proved its powerfulness in capturing a high abstract conceptual model of the enterprise. DEMO models can be used in process re-engineering as well in enterprise engineering. However, using DEMO models in a stage prior to requirement engineering for developing information system is still in progress. A few frameworks are developed for

developing enterprise information system based on DEMO. Nevertheless, very few real world case studies applied the frameworks. Therefore, more real world case studies are needed to enhance and justify those frameworks. For example, DEMO processors that compile and execute the DEMO models have been developed (Kervel, Dietz, John, Meeuwen, & Zijlstra, 2012).

On the other hand,i* model is a previously established framework in requirement engineering for developing information systems (Pandey, Suman, & Ramani, 2010). By comparing i* with DEMO, we can thus formulate a framework for developing information system based on DEMO models.

i* (I STAR) FRAMEWORK

The i* (I star) framework is one of the most widely adopted modeling approaches by several communities (e.g., requirements engineering, business process reengineering, organizational impacts analysis and software process modeling). It is a goal- and agent-oriented modeling and reasoning framework that defines models that describe the systems with the environments in terms of intentional dependencies among strategic actors (Pandey, Suman, & Ramani, 2010). It is used for comparing many different scenarios for the same system by changing the dependencies between the agents (Liu, et al., 2014). There are two different models: (1) the strategic dependency (SD) describes information about dependencies and (2) the strategic rationale (SR) defines the actor details. The SR model complements the information provided by the SD model by exploiting internal details of the strategic actors to describe how the dependencies are accomplished.

Figure 4-2 shows SD of the same simple case that represented in CM model in DEMO. Actors are represented by circles. Customer asks the solution provider an IT solution. Here, the solution represented by oval to show that it is goal type. The solution provider asks the

payment from the customer. Payment is represented by a rectangle because it is considered a resource type.

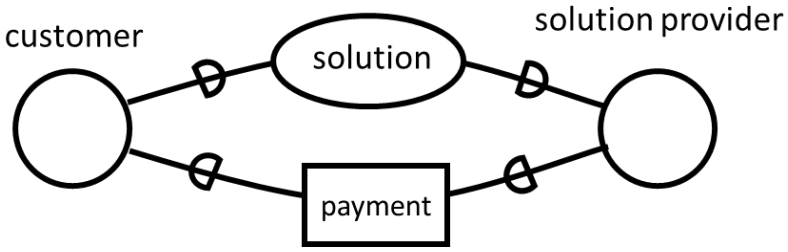


Figure 4-2: SD of a simple case

In i*, there are four types of dependencies that are characterized according to the dependum. The dependum can be a soft-goal, a goal, a task or a resource. Soft-goals are associated with non-functional requirements (NFRs), while goals, tasks, and resources are associated with system functionalities (Castro, et al., 2012).

4.2 Comparison Case Study

The following passage describes a case study modeled by both DEMO and i*. Those models will be used later for specifying the requirements to develop information system. This section consists of four parts. First is a textual description of the case study. Second is explaining the objectives of developing the information system. Third is the DEMO model with its explanation. Fourth is i* model with its explanation.

BACKGROUND OF THE SELECTED CASE

SMA offers its customers IT solutions by developing software based on the customer's needs or by providing consultation. SMA is a project-based company. Every project belongs to one client who may have more than one project with the company. The employees do not form a structure based on the organization chart; rather they are flexible based on the projects they have. This flexibility allows the company to respond quickly to the changes in the market.

In every project, there is one project manager leading a team of developers. The project manager is responsible for planning the project, as well following it to completion.

Based on the project, different person from project manager would be responsible for delivering the product to the customer. This person may also be responsible for taking care of the payment from the customer. Otherwise, the project manager does the delivery and receives the payment from the customer. When a new project arrives, the project manager begins by planning the project.

As a result of the planning, a list of tasks with their schedules is made. After breaking down the project into tasks, the project manager assigns the tasks to the developers. During the execution of the project, new tasks may pop up. Therefore, every developer may assign a new task to himself/herself or assign it to the other developers. All the tasks must be recorded in the information system to be developed. This is very important to follow up the completion of each task.

The project manager controls the completion of the tasks every week. The project manager then looks at the completed tasks and the remained tasks. He/she reassigns the tasks from developers with work overload, to those who have less work. This provides a work balance for every developer, to allow efficient project completion. At the same time, the manager may control the execution of the tasks by prioritizing them according to their importance.

Employees receive their salaries based on their work time. Therefore, they record the time for completing every task they do. In addition, at the end of the month, the accountant calculates the work time for each employee. For each employee, there is a specific hourly salary rate. Based on this rate, the actual payment is calculated. The salary is the sum of work time multiplied by the salary rate plus the reward.

Employees may ask for bonuses or other rewards. This is done after evaluating their performance. The project manager analyzes the performance of all employees based on their task completion rate. Moreover, based on the performance analysis, the salary rate may increase, or a reward for a particular project may be given.

Employees are free to choose their time to work, i.e., day or night, as long as the projects are proceeding as scheduled. This flexibility gives them responsibility for their time.

To keep the level of the skills in the company up to date, SMA frequently hires new highly qualified developers.

DEMO MODEL

Based on the description in the previous two paragraphs, the DEMO model of SMA can be constructed as follows. Because of pages limitations, only the organization construction diagram (OCD) of the construction model (CM) will be detailed. The unit of business service of SMA is the provision of an IT solution to the customer. Therefore, the first transaction to be identified is the (T1) project completion. The customer (CA1) initiates this transaction by requesting an SMA employee to provide an IT solution. This employee will be called the project completer (A1). A1 initiates three transactions: (T2) project fee payment, project planning (T3) and task completion (T4). It must be said that T4 cannot be requested before the plan of the project is done. Since the project manager controls the execution of the plan, then he or she is taking the role of task manager (A5). This actor initiates and executes periodically (every week) the transaction task management (T5). The execution of this transaction leads to change the project plan. Therefore, plan revision transaction is needed (T6).

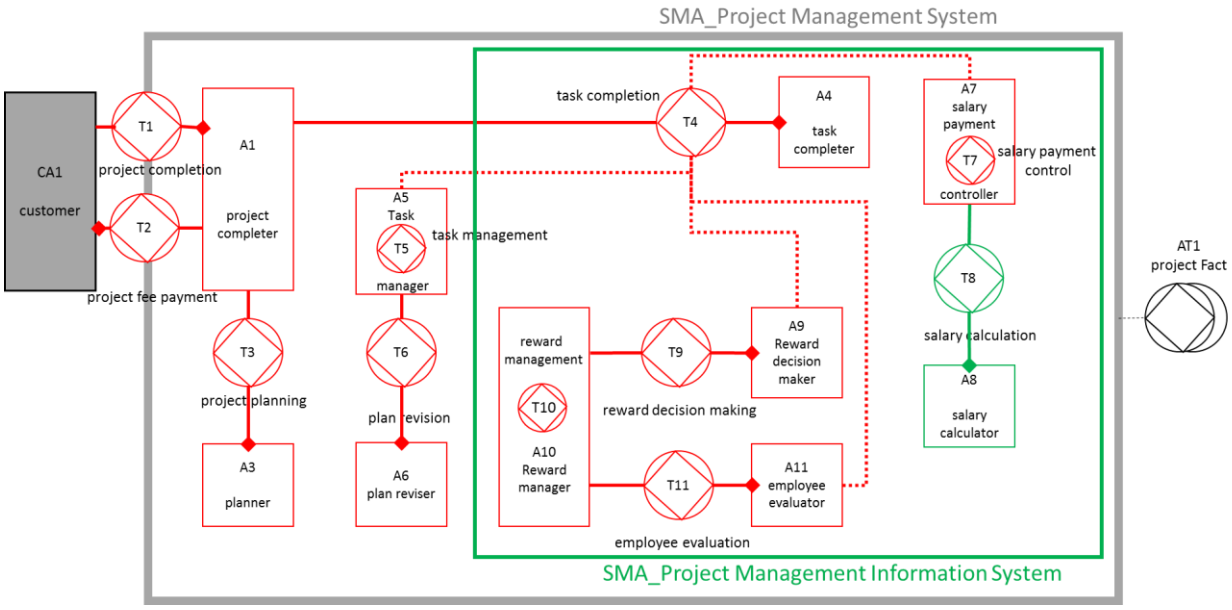


Figure 4-3: Actor transaction diagram of SMA

To model the salary and the reward that are mentioned in the description, salary payment control (T7) and reward management (T10) transactions are needed. To execute T7 we need to calculate the salary. Because there is no original fact in T8 (only calculation), then it is infological transaction (green). To execute T10, we need two sub-transactions, reward decision making (T9) and employee evaluation (T11). Based on the previous paragraph that describes the objectives of the information system to be developed, the scope of the information system is shown by a green rectangle. The customer has no

relationships with the information system. This is because SMA prefers always to communicate with the customer directly to form good human relationships. The OCD of the

CM is shown in Figure 4-3.

I* MODEL

Because the objective of the model is to develop an information system, then only the actors who are involved in the system will be modeled. In i*, unlike in DEMO, we starts by modeling the actors. There are three actors: manager, employee and accountant. The manager

has a dependency relationship with the employee by asking him/her to achieve a particular task. The dependency is of the task type because it is a task. The employee has two dependency relationships: salary and reward. Because they refer to the money to be given from the accountant to the employees, both dependencies are of the resource type. To give the reward, a performance analysis is needed. Therefore, the accountant depends on the manager to do the performance analysis for the employee before giving the reward. This is a resource type dependency. The strategic dependency (SD) model is shown in Figure 4-4.

For each actor in the SD, there is a rational dependency (RD) model. In this research, only one RD will be modeled. In Figure 4-5, the RD for the manager is shown. The RD shows the internal tasks that the actor performs to respond to the external dependencies of the other actors. In SMA, the manager controls the tasks for each employee. The task can be decomposed into two tasks: assigning a task and releasing a task. These two tasks influence the soft-goal balanced load. At the same time, controlling the tasks is required for evaluating the achievement of an employee. The result of the evaluation is the performance analysis, which is delivered to the accountant.

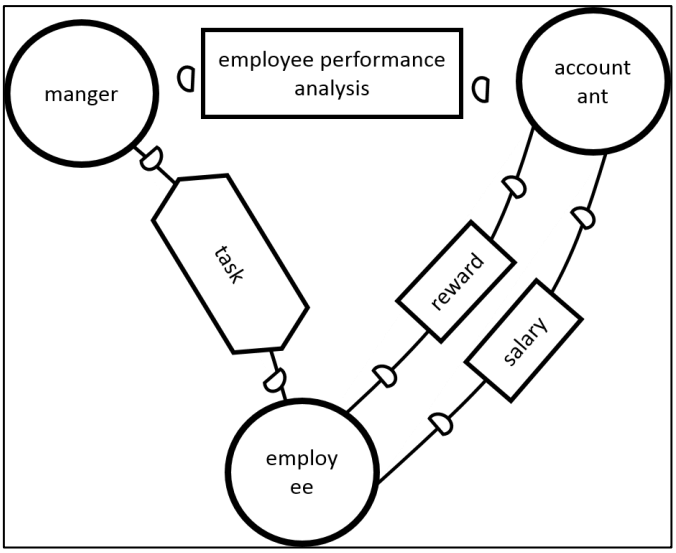


Figure 4-4: Strategic dependency model of SMA.

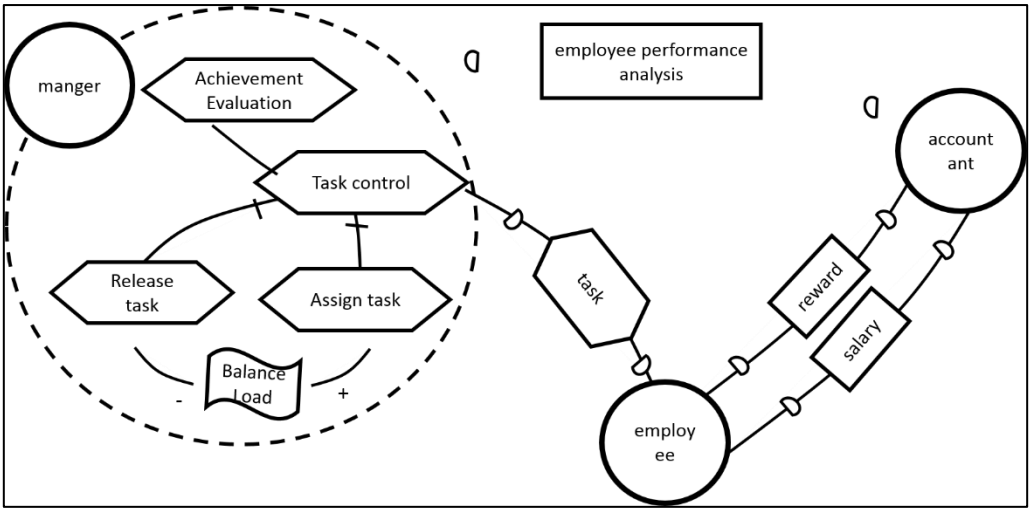


Figure 4-5: Rational dependency model of SMA.

4.3 Discussion

First, both DEMO and i*are social modeling methodologies. In their models, enterprise consists of actors (actor role in DEMO and agent in i*) that interact with each other through relationships (transaction in DEMO and dependency in i*). However, the concept of actors and relationships are different. In i*, humans are modeled by an actor with concrete names, for example, manager, employee and accountant. However, in DEMO, it is more abstract such as planner and task controller. After presenting these two models to the stakeholders, it was more easy for them to understand i* model than DEMO. This is because i* uses more concrete roles that are familiar to the stakeholders. The following table matches the actor role in DEMO with the actor in i*.

Table 4-1: Actors in DEMO and i*.

DEMO actor role	i* actor
task manager, plan reviser, reward manager, and employee evaluator	Manager
Task completer	Employee
Salary payment controller, salary calculator, and reward decision maker	Accountant

From the Table 4-1, we can see that every actor in i* can be broken down into actor roles in DEMO. Therefore, a composite actor role can be used in DEMO to make it easier to be understood. Every composite actor role can be later decomposed into its elementary actor roles. This facilitates discussing the model with the stockholders.

Second, there is a difference between a transaction in DEMO and a dependency in i*. In DEMO, transactions are divided into ontological, infological and datalogical categories according to the abstraction level. The differences between them are clear by definition. Whereas in i*, the dependency is divided into goal, resource, task and soft-goal. However, the difference between goal, resource and task is not clear. This looseness may not be important in some situations. However, in others situations, such as model-driven development, it is very important (Oriol, Marco, & Franch, 2014). For example, in our model, both reward transaction and salary transaction are modeled in i* as a resource. However, task completion transaction in DEMO is modeled in i* as a task. Therefore, no automatic transformation between transaction in DEMO into i* could be done.

Third, i* is capable of capturing the rational aspect of the system by RD model. Agent in i* could be decomposed into rational elements. This facilitates the implementation by automating them. However, DEMO is considering only the social part of the system. This makes it difficult for implementing the system in later stages.

Fourth, i* models soft-goals. This is useful for modeling the nonfunctional requirements in the early stages. Service quality and service speeds are examples of nonfunctional requirements. However, DEMO considers only the functional requirements. Even DEMO doesn’t capture the nonfunctional requirements, one way to do that is to consider any nonfunctional requirement as a functional one and modeled it in a transaction.

Fifth, DEMO has four perspective models (construction, process, fact and action) that captures a holistic view of the enterprise. This is very important in developing any information system (Figueiredo, Souza, Pereira, Prikladnicki, & Audy, 2014). However, i* does not have equivalent to fact and action models of DEMO.

4.4 Conclusion

In this chapter, a comparison between two modelling methodologies named DEMO and i* in modelling enterprise as a prior stage of requirements analysis is made. By the comparison between DEMO and i*, both of them are social modeling methodologies. They focus on human and human interaction in their modeling. DEMO is implementation independent. Therefore, the DEMO model does not change before or after implementing any IT solutions. However, i* is implementation dependent methodology. DEMO provide more formal and rigour model of the enterprise. That makes it a good potential modelling methodology to understand the enterprise before implementing any IT solution. On the other hand, i* allows us to capture both the rational and the social aspect of the enterprise. The rational aspect is important in developing any information system. Therefore, DEMO should be extended to capture the non-social aspect, too.

There are some frameworks for developing information system based on i*. Since we showed the strength of DEMO, the next step is to develop such a framework like the i* has. Another point to be considered in the future is extending DEMO to capture the rational aspect of the enterprise like i*. This is important for developing information systems.

CHAPTER 5: DEMO SIMULATION METHODOLOGY

5.1 Introduction

Enterprises are growing in complexity due to the existence of many business processes that form an interwoven network of business transactions. To design and re-engineer such an enterprise, a conceptual model of the enterprise is needed. In recent decades, many modeling methodologies have been developed. Among those methodologies is DEMO. Design and Engineering Methodology for Organizations (DEMO) is a methodology for the design and engineering of organizations. DEMO is a concise and coherent model that illustrates the essence of an organization (Dietz, Enterprise Ontology Theory and Methodology, 2006). However, DEMO lacks tools that support the simulation of its models. DEMO Simulation provides a powerful tool to validate the proposed DEMO model compared to the real world by running, debugging and analysing the model. Deadlocks or any unpredicted roots can be discovered during the simulation. Furthermore, simulating DEMO models may answer "what if" questions, which may be a useful tool for re-engineering the enterprise. Additionally, Petri net is a simple modelling language used to model and analyse concurrent systems. Its simplicity and simulation capability make it appealing for many other modeling languages. They transfer their model into PN to utilize its features, e.g., Activity Diagram and BPMN. However, the usefulness of the simulation model depends on the quality of the original conceptual model like AM or BPMN. And if those models don't represent the ontology of the enterprise, then the quality of the simulation models will be low. Therefore, an conceptual ontology model is needed as a base to do the simulation. We call it here ontology based simulation. By analyzing PN and DEMO models, the similarity between the two is clear. The concepts of Fact and Act in DEMO model can be perfectly mapped to the concepts of Place and Transition in PN. This similarity creates the potential to map DEMO to PN, which is

useful not only for simulating DEMO models but also for utilizing the richness of research and analysis that are conducted on PN, as well as other tools that have been developed for those purposes. Therefore, in this paper, a transformation methodology from DEMO to Coloured Petri Net is introduced. The transformation mainly focuses on the Process Model (PM) of DEMO, because PM describes the dynamics of the enterprise as a workflow and is used to create the Basic Petri Net. However, the State Model (SM), and the Action Model (AM) of DEMO are also used to define the Color Sets, Guards, Variables and Expressions in CPN. The potential benefits from transforming DEMO into CPN can be summarized as follows: first, CPN is based on Petri net, which can be used for simulation. Therefore, the transferred model can be used to simulate the DEMO models. Second, CPN is an old and simple process modeling language that provides a lot of expertise and tools that can be used for analysis of its models. Third, CPN has a mathematical representation that can lead to interesting research that analyses DEMO models mathematically. Reasons for choosing Coloured Petri Net include its ability to capture the cardinality in DEMO and its ability to program the business rules in the AM of DEMO by constructing the Guards and Expressions in CPN. Therefore, the main contribution of this paper is the transformation of the DEMO model into CPN and showing the validity of our transformational approach by implementing one case study. The remainder of this paper is organized as follows. First, we address the literature review, covering the features of the DEMO Model and Petri Net. Then, the transforming methodology will be introduced. To validate the proposed method, a case study will follow this methodology. Finally, in the last chapter, the conclusion and the discussion are addressed with the results and the future work.

5.2 Petri Net

A Petri Net is one of the modeling languages for the description of distributed systems. The modeling languages of Petri Net consist of transitions represented by a rectangle, places

represented by circles, and edges that connect the transitions with the places. Places act like a pre/post condition for the transition. (Marwan, Rohr, & Heiner, 2012).

Definition 1. A Petri net is a triple N = (P, T, F) where:

P and T are disjoint finite sets of places and transitions, respectively.

$F \subset (P \times T) \cup (T \times P)$ is a set of arcs (or flow relations).

Petri Net has the following features (Girault & Valk, 2003) (Heiner, Liu, Rohr , & Schwarick, 2012):

Representations: Petri Net has both a graphical and mathematical representation that can be used for modelling and analysing the systems;

Verification: There are many algorithms for verifying the model as well as tools for analysing Petri Net models, and these algorithms are supported by many powerful computer tools;

Hierarchy: Petri Net has the ability for form abstractions and hierarchical designs, which is a crucial factor for the effective design of complex systems. There are many mechanisms for abstraction and refinement that can be used for modelling systems; Expertise: Because Petri Nets have been used in many different application areas, there is a high degree of expertise in the modelling field. Some examples would be manufacturing, workflow management, telecommunications and biology;

Varity: There are different variants of Petri Net models that have been developed to suit different applications, such as Coloured Petri Net (CPN) and Stochastic Petri Net;

Simulation: Petri Net can be simulated, and it has many tools for simulation. Therefore, it is possible to perform many experiments using the model and then analyse the results;

Demonstration: Above all, the simulation ability of Petri Net makes it useful as a good demonstration for the stakeholders to achieve a common understanding about the model.

There is a subclass of Petri net called workflow nets (WF nets) that is used for modelling and simulating business process and workflow. WF nets can be defined as follow:

Definition 2. (Aalst, et al., 2011) A Petri net PN= (P, T, F) is a WF-net if and only if:

There is one source place $i \in P$ such that $\bullet i = \emptyset$

There is one sink place $o \in P$ such that $o\bullet = \emptyset$

Every node $x \in P \cup T$ is on a path from i to o

, where $\bullet i$ is the set of transitions sharing i as output place, and $o\bullet$ e is the set of transitions sharing o as input place. And there are many concepts and criteria that have been developed for the purpose of verification of WF nets. One of the most important criteria is that of soundness.

Definition 3. (Aalst, et al., 2011) A WF-net is sound if and only if:

For every state M reachable from state i, there exists a firing sequence leading from state M to state o.

State o is the only state reachable from state i with at least one token in place o.

There are no dead transitions (transition that can never fire)

In this paper, we use Coloured Petri Net to capture the cardinality in DEMO and the business rules in Action Model.

Definition 4. A Coloured Petri Net is a tuple $N = (P, T, F, \Sigma, C, N, E, G, I)$ where:

P is a set of places.

T is a set of transitions.

F is a set of arcs

Σ is a set of color sets defined within CPN model. This set contains all possible colors, operations, and functions used within CPN.

C is a color function. It maps places in P into colors in Σ.

N is a node function. It maps F into $P \times T \cup T \times P$.

E is an arc expression function. It maps each arc $f \in F$ into the expression e. The input and output types of the arc expressions must correspond to the type of nodes the arc connected to.

G is a guard function. It maps each transition $t \in T$ into guard expression g. The output of the guard expression should evaluate to Boolean value true or false.

I is an initialization function. It maps each place p into an initialization expression i. The initialization expression must evaluate to a multiset of tokens with a color corresponding to the color of the place C(p).

Despite all of these features of Petri Net, many other modeling methods are used to describe the system, such as AD (Activity Diagram in UML) and BPMN (Business Process Modelling Notation), even they do not have a tool for a simulation like Petri Net. The reason for this discrepancy is that these types of modeling methodologies have more representation elements using graphical representations, which can be easily understood by stakeholders, unlike Petri Net, which has only transitions and places (Weske, 2012).

In fact, many researchers have proposed a transformation methodology from different models, such as AD and BPMN into Petri Net, as described below. Furthermore, PN lacks the

ontology concept in modeling. Without the ontological concept, models could be very complex and lack the consistency and the completeness that DEMO has. From the previous paragraph, we can conclude that DEMO and Petri Net together can construct a perfect methodology for modeling and simulating the enterprise.

5.3 Transforming BP Models into Petri Nets

There are several studies on transforming different business modeling languages to PN. For example, AD is a diagram that can express the most desirable routing constructs, but it has no defined semantics that is used for workflow modeling. A transformation of AD into Petri Net allows for model checking for verification and validation purposes. Other studies have been performed for the purpose of evaluating non-functional parameters of a software system in the design stages using Generalized Stochastic Petri Net that has been transferred from the AD model (Eshuis & Wieringa, 2002) (Motameni, Movaghar, & Fadavi Amiri, 2007) (Staines, 2008).

Another research has been conducted for transforming BPMN into Petri Net to check the semantic correctness of the models statistically (Dijkman, Dumas, & Ouyang, 2008).

Based on the previous research, we can see that many studies have developed methodologies for transforming a business process model into PN. However, DEMO dose not has this transformation yet, which is introduced in this research. Previous research proposed a simulation of DEMO using the Standard Petri Net. However, the full transformation that includes all of the transaction patterns is not developed. In this research, a full transformation methodology from the Process Model of DEMO into PN is introduced for the three transaction patterns (basic, standard and complete). In addition, Coloured Petri Net is proposed for describing the cardinality of the DEMO model. Furthermore, the business rules

that are described in the Action Model of DEMO can be programmed in Coloured Petri Net (Barjis, 2007).

5.4 Transformation Methodology

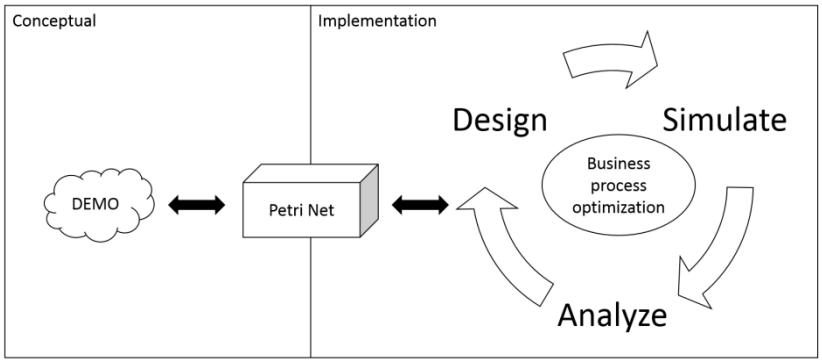


Figure 5-1: Business Process Optimization based on DEMO and CPN.

In Figure 5-1, the conceptual idea of the transformation is presented. After creating the CPN model from the DEMO model, a configuration is needed to specify the instances that are required for simulation, as shown in the design phase. After the configuration, the simulation is conducted. CPN can be simulated interactively or automatically. The interactive simulation is a single-step debugging. This method is used to validate the model. In addition, this method was used in the case study presented in this paper. It provides a way to "walk through" or investigate the different scenarios in detail and determine if the model works as expected. The second one is the automatic method, which is used for performance analysis.

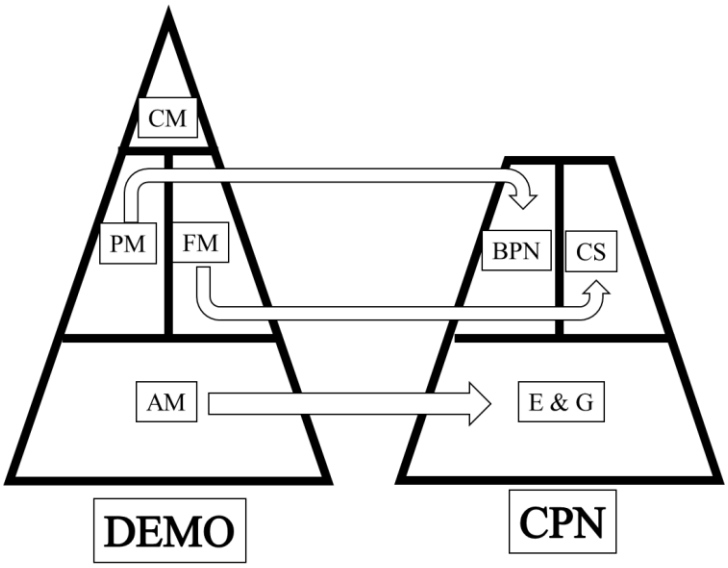


Figure 5-2: Transforming DEMO to CPN

In this research, each aspect of the DEMO model will be mapped to CPN (except CM). The DEMO model consists of four aspect models: a Construction Model (CM), Process Model (PM), State Model (SM) and Action Model (AM). The CM provides a general view of the enterprise by showing the transactions related to the actor roles. The PM provides more details about the process steps between the transactions. Therefore, the Basic Petri Net (BPN) can be constructed based on the PM. The BPN consists of Place, Transition, and Edge. The SM illustrates the object classes with their properties. These classes and their properties will form the Color Sets in the CPN. Finally, the AM presents the business rules. These business rules govern the actions between the process steps and will be created in the CPN using the Transition Guards (G) and Edge Expressions (E), which is shown in Figure 5-2. To illustrate the transformation from DEMO to CPN, the CPN model of the transaction will be presented for the standard transaction pattern and for the complete transaction pattern.

STANDARD TRANSACTION PATTERN

Petri net has three basic elements place, transition, and edge. Initiate, C-fact, Discussion status and P-fact in DEMO will be replaced by Place in Petri Net because the Fact and Status

in DEMO represent a particular status of an instance. This concept is perfectly matched with the concept of Place in Petri Net. The C-fact and P-fact in DEMO have no difference in Petri Net because both of them are replaced by a Place. The Acts in DEMO (both C-act and P-act) can be replaced by Transitions in Petri Net because the concept of an Act in DEMO represents a change in the status of an instance, which can be matched perfectly by the concept of a Transition in Petri Net, as shown in Figure 5-3. All of the types of links in DEMO will be replaced by Edges in Petri Net. In the case of a Causal link (there is no arrow), the arrow will be from the Transition to the Place by default. When there is more than one (In or Out) link for one transition in Petri Net, it is handled as an And Joint. However, in DEMO, there might be an And Joint or an XOR joint. To solve this issue, a composite transition is introduced, which is illustrated in Figure 5-4. Petri net has three basic elements place, transition, and edge. Initiate, C-fact, Discussion status and P-fact in DEMO will be replaced by Place in Petri Net because the Fact and Status in DEMO represent a particular status of an instance. This concept is perfectly matched with the concept of Place in Petri Net. The C-fact and P-fact in DEMO have no difference in Petri Net because both of them are replaced by a Place. The Acts in DEMO (both C-act and P-act) can be replaced by Transitions in Petri Net because the concept of an Act in DEMO represents a change in the status of an instance, which can be matched perfectly by the concept of a Transition in Petri Net, as shown in Figure 5-3. All of the types of links in DEMO will be replaced by Edges in Petri Net. In the case of a Causal link (there is no arrow), the arrow will be from the Transition to the Place by default. When there is more than one (In or Out) link for one transition in Petri Net, it is handled as an And Joint. However, in DEMO, there might be an And Joint or an XOR joint. To solve this issue, a composite transition is introduced, which is illustrated in Figure 3-2. In the Standard Transaction Pattern, Rq and St Transitions have two input arrow. In addition, they have to act as XOR junction. Therefore, they are represented by

a composite transition that consists of one place and three transitions. The Petri Net model of the Standard Transaction Pattern is shown in Figure 5-5.

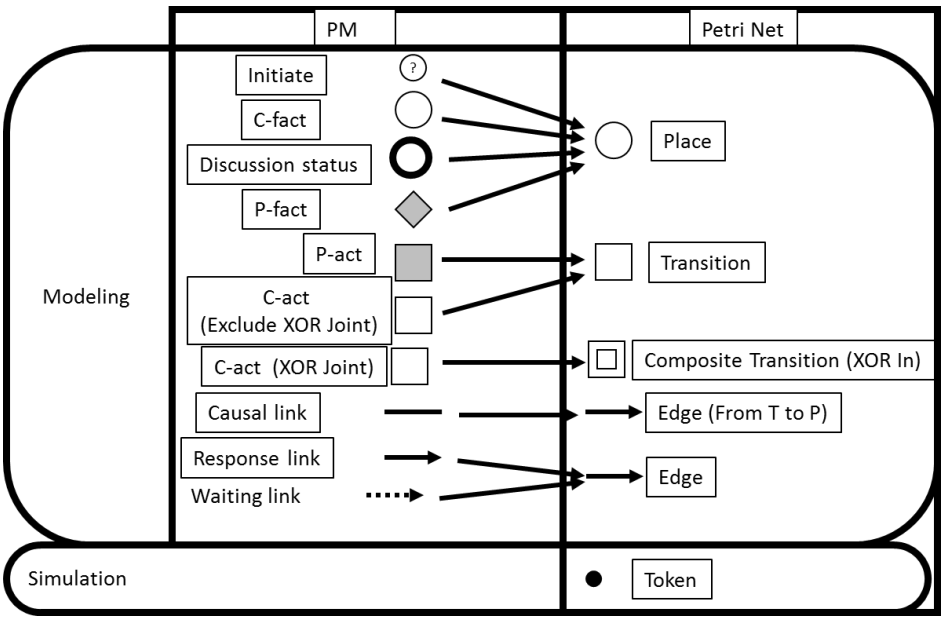


Figure 5-3: Elements mapping from PM of DEMO to PN

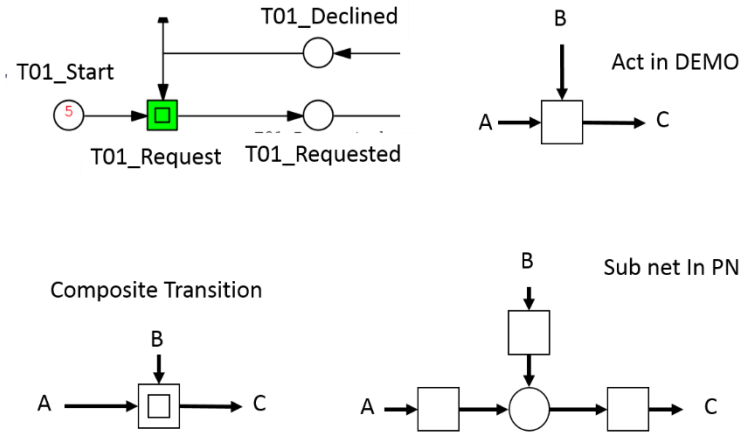


Figure 5-4: XOR joint In PN

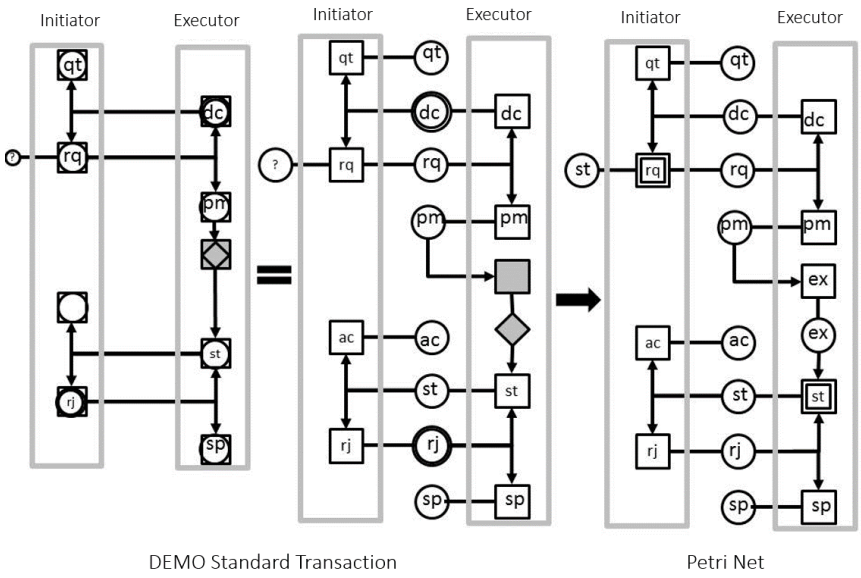


Figure 5-5: Petri Net model of the Standard Pattern Transaction

COMPLETE TRANSACTION PATTERN

In the Complete Transaction Pattern, the revoked process needs to be added to each request, promise, state and accept. For each revoke, the revoke starts by requesting the revoke (one transition and one place). If the revoke is allowed, then the token (which represent an instance) will be revoked and sent to the previous process.

CONFIGURING THE CPN MODEL OF DEMO

After transferring each transaction to the Petri Net model, the links between the transactions can be added to the Petri Net model. For instance, if there is a link between promise at T1 and a request at T2, a link is made from the promise transition of T1 to the start of T2. In this step, the unnecessary process can be deleted (reject and decline for example).

If the purpose of the simulation is to compare many different possible flows, then many different models (by adding or deleting the reject, decline, and revoke) can be constructed and compared. After completing the model, a set of color sets can be defined. These color set

should represent the properties that are to be measured in the simulation results. For instance, if we want to measure the cost, a cost color set can be added to the token in the Coloured Petri Net. After the simulation, the result for the cost can be analyzed.

5.5 Case Study

The following passage describes the case study that will be used as an example for validating the transformation methodology. This example has been taken from analyzing a typical fast food restaurant in Syria. The description is as follows:

This passage is a description of a typical sandwich restaurant in Syria. In this paper, it will be referred as TSR (Typical Sandwich Restaurant). The restaurant sells many different types of sandwiches (Falafel, Shawarma ...). Customers can customize their order by specifying the spices and the dressing for their sandwiches. Customers come to the cash register where they choose their order from the menu, and if they want, they can specify customized sandwiches according to their taste. The payment is performed immediately at the cash register when they order. After they have paid, they receive a receipt that has all of the details of their order, and then, the customer goes to the chef and gives him the receipt. Some restaurants have automated this process in such way that the order is automatically shown on display in front of the chef. In the automated method, the customer receives a receipt that has only the order number. Each order can contain one or more sandwiches. Each sandwich is made separately from the others. Therefore, the order can be performed by only one worker or more than one according to their availability. All of the workers can do all of the tasks, including taking orders, making sandwiches and giving the finished order to the customers. Assigning the tasks to workers is performed by the manager who needs to always monitor the entire process and to adjust to the situation, which means that if there are many customers waiting for someone to take their order, the manager will ask more than one worker to take orders. However, if there is one large order (more than 10 sandwiches), then he will assign more than

one workers to fulfill this order. After completing all of the requested sandwiches, a worker collects them together, puts them in a package and gives them to the customer. To respond immediately to the situation of the needed number of workers at each section, the manager needs to construct a dashboard that displays the current state of each section in one model. In this dashboard, the number of waiting customers and the reason for their wait (for example, they are waiting for their sandwiches to be made, or they are waiting for someone to take their orders…) must be displayed. The status of each order has to be displayed (for example, how many sandwiches have been made and how many sandwiches have not yet been made). This information should be displayed in one model that alloys the manager to understand the situation and respond to it as soon as possible. The small changes in the work procedures should not affect the model; otherwise, for each new procedure, we may need a new model, which could cost a lot.

Based on the description, we can construct an ATD (Action Transaction Diagram) using DEMO, which is useful for understanding the ontological aspects of the restaurant. ATD is the basic diagram of DEMO that shows the ontological transactions linked to the business roles: who are the initiators and executors for these transactions. The ATD of the TSR is shown in Figure 5-6. The first transaction is (T1) purchase completion. The customer is the initiator of this transaction (order sandwiches). Because the customer is considered an external actor role, it is shaded CA1 (Composite Actor Role). The executor of T1 is the A1 receptionist who takes the order. The executor of any transaction is always differentiated by the black dot on the link to its transaction. The same actor role A1 is the initiator for the second transaction T2 payment because the receptionist asks the customer to pay, and the customer is the executor of T2 (has the black dot). The third transaction is T3 making sandwiches. T3 is an internal transaction because both the initiator A1 and the executor A2

are actor roles of the restaurant. Based on this model, it is clear that the automation of one process or a small change in the workflow will have no influence on this model.

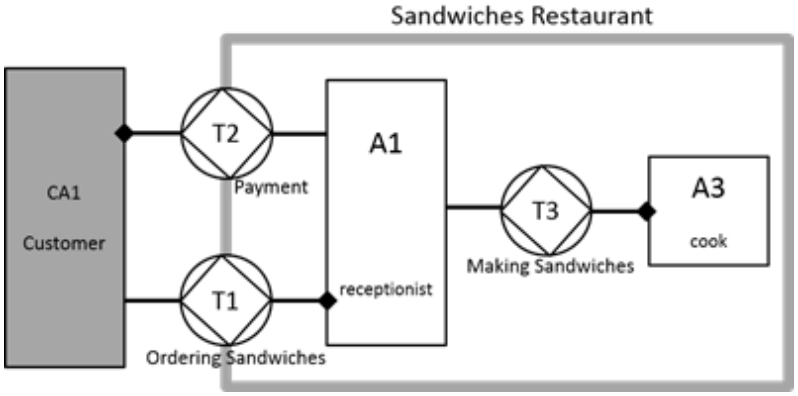


Figure 5-6: Actor Transaction Diagram of TSR

ATD does not show the execution sequence of the transactions. The sequence of transactions is illustrated in the PSD (Process Structure Diagram). The process starts by requesting the transaction T1 by CA1. After promising T1 by A1, T2 is requested. When the payment transaction T2 is accepted by A1, T3 is requested by A1 (the cardinality number 1..n means the number of sandwiches is more than or equal to 1 and finite). Finally, when all of the sandwiches are made, T1 can be executed after accepting T3 by A1 and stated to the customer CA1. In Figure 4-2, there are three links among transactions. These links will be represented by red in Petri Net. To get the CPN model of TSR, first, each transaction will be replaced by the suitable pattern that is explained in the previous section. Then the configuration needs to be set as it is explained in 3.3. The configuration is explained in the following passage.

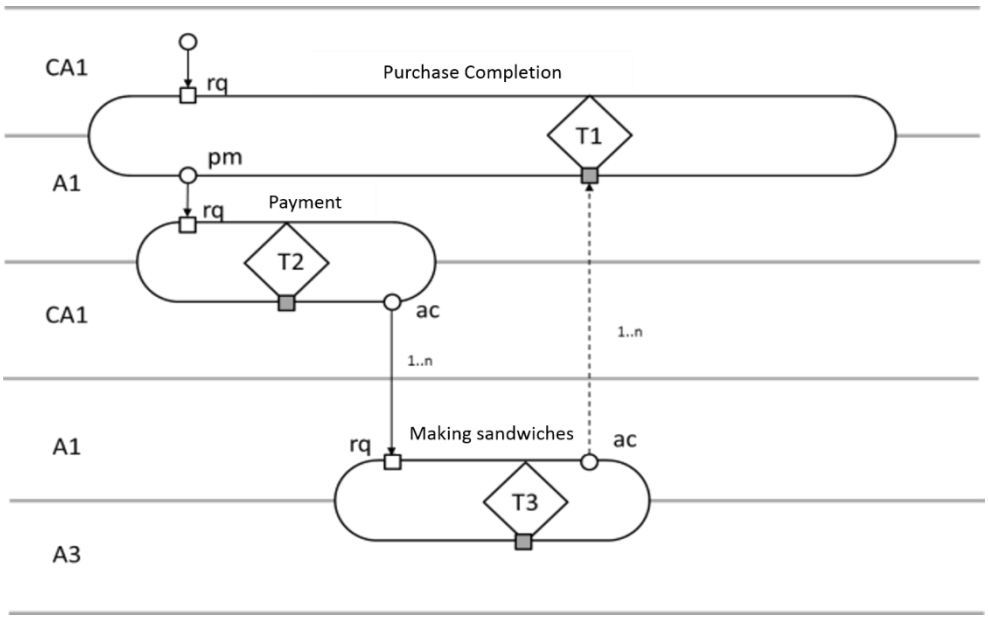


Figure 5-7: Process Structure Diagram of TSR

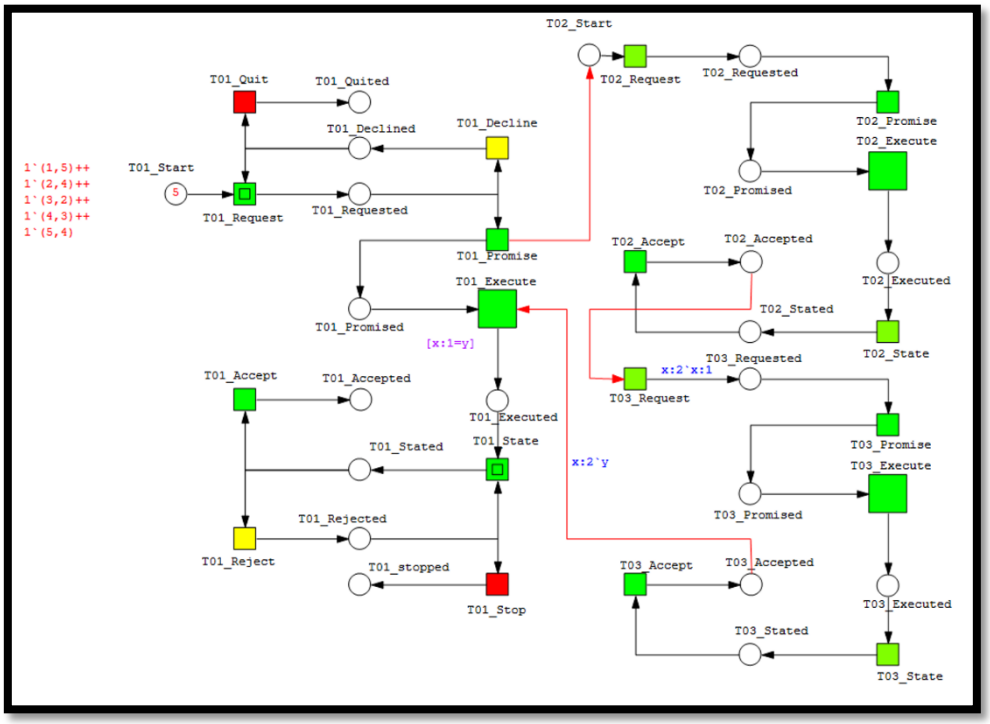


Figure 5-8: Coloured Petri Net model of TSR

5.6 Discussion and Conclusion

DISCUSSION

In Figure 5-8, the colored Petri Net model of the TSR is represented. Reject and decline at T2 and T3 have been deleted because when the customer orders the sandwiches, they already know about the prices, and there is no room for declining or rejecting the payment. The same is true for the making of the sandwiches: there is no reason to not make the sandwiches. If the purpose of the simulation is to study the availability of raw materials, it is possible not to be able to make the sandwiches, and the decline process must be added. In addition, the Quit place must be connected to the revoke of promise at T1, which is necessary to roll back the token to T1. Two color sets have been defined for the tokens. The first color set represents the ID of the order, which is necessary to ensure that all of the sandwiches go to the same order. The second color set represents the number of the sandwiches, which is the cardinality in T3. For example, if there is one order with three sandwiches, then the token of the order will wait at the promised place until its token (the sandwiches) are accepted. After all of the sandwiches are accepted, then the order can proceed to the execution transition.

Another important point is the initiation of T3. The case study shows that T3 starts after accepting the payment at T2. However, it is not necessary to wait for the payment. It can be started directly after promising at T1. In this case, the acceptance of T2 will be linked to the execution of T1. By analyzing the resulted Petri Net model, the model fulfills the three conditions of WF-net. There is one source place corresponds to the initial state (T1 Start). And there are three possible sink places (T1_Accepted, T1_Quitted, and T1_Stopped). And each node is on the path from the source place to one of the three sink places. This could be easily seen by drawing the State Space Graph in CPN Tools (Jensen & Kristensen, 2009). State Space Graph represents all the possible state (marking) that the system can be. Moreover, to test the soundness of the model, we applied the Corollary 1 (Aalst, A class of

Petri nets for modeling and analyzing business processes, 1995). Corollary 1 says "A BP-net PN is sound if and only if ((PN)¯,i) is live and bounded." ((PN)¯,i) is the model and adding transitions to each sink place to the source place. Therefore, three transitions have been added to the model. Then by analyzing the new model using the CPN tools, it shows that it is live and bound. This means the original model is sound.

CONCLUSION

In this research, we propose a methodology for transforming models from DEMO to Coloured Petri Net (CPN). The transformation of PM is formal and can be programmed to any tool for automatic transformation. The transformation is mainly based on the PSD of DEMO; however, business rules in the Action Model (AM) are included as well when the links in CPN are programmed. It is important to specify the cardinality in PSD. In PSD, the cardinality is not always one to one. It is possible to have a one-to-n token, which means that one token from the first transaction will produce n tokens in the second transaction. For example, one order may have many sandwiches. The number of tokens depends on the data value that the token has. A case study of a typical sandwich restaurant was used to validate this methodology. The CPN model of the TSR was capable of capturing the standard pattern transaction as well as the basic pattern. Moreover, the cardinality in DEMO model of TSR (number of sandwiches) was represented in the CPN model of TSR. This shows that this transformation methodology overcomes the shortage of previous research (capture all the patterns and the cardinality). And the case study shows that the methodology is valid for capturing those properties.

CHAPTER 6: CASE STUDY

In this chapter, we show the development of an IS for a company called SMA based on the proposed framework in chapter four.

6.1 Background of the Selected Case:

SMA is a leading information technology company. Using its global network delivery model, innovation network, and solution accelerators, SMA helps global organizations address their business challenges.

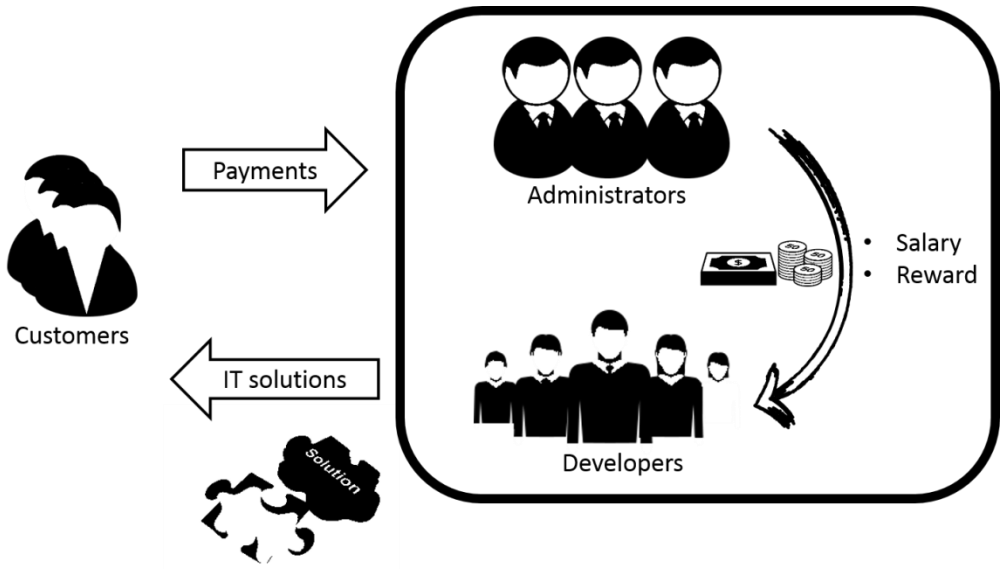


Figure 6-1: SMA Illustration

Figure 6-1 shows the main workflow in SMA. SMA offers its customers IT solutions by developing software based on the customer's needs or by providing consultation. SMA is a project-based company. Every project belongs to one client—who may have more than one project with the company. The employees do not form a structure based on an organizational chart; instead, they are flexible based on the projects they are currently working on. This flexibility allows the company to respond quickly to changes in the market. Therefore, each project has one project manager leading a team of developers. The project manager is

responsible for planning the project as well as following it to completion. Based on the project, there might be an employee who is responsible for delivering the product to the customer. This employee may also be responsible for handling payments from the customer. Otherwise, the project manager delivers and receives payments from the customer. When a new project arrives, the project manager begins by planning the project. The result of this planning is a list of scheduled tasks. After breaking down the project into tasks, the project manager assigns those tasks to the developers. During project execution, new tasks may arise. Therefore, every developer may assign a new task to himself/herself or assign it to other developers. All the tasks must be recorded in the information system to be developed. This is critical for following up on the completion of each task. Moreover, the performance analysis and salary of each employee are based on performance—on the results of completing these tasks. The project manager monitors task completion every week, looking at the completed tasks and the remaining tasks, and reassigning tasks from developers with heavy workloads to those who have less work. This balances the work for every developer and improves project efficiency. At the same time, the manager may control task execution by prioritizing tasks according to their importance. Employees receive their salaries based on their working time. Therefore, they record the time for completing every task they perform. At the end of the month, an accountant calculates the work time for each employee. Each employee has a specific hourly salary rate. Based on this rate, the actual payment is calculated. The salary is the sum of work time multiplied by the salary rate plus any reward. Employees may ask for bonuses or other rewards. These are awarded (or withheld) after evaluating their performance. The project manager analyzes the performance of all employees based on their task completion rate. Moreover, based on the performance analysis, an employees' salary rate may increase, or the employee may be given a bonus for a particular project. Employees are free to choose the times at which they work, i.e., day or night, as long as the projects are

proceeding as scheduled. This flexibility gives them responsibility for their own time. To keep the level of the skills in the company up to date, the SMA frequently hires new highly qualified developers.

6.2 SMA DEMO Model:

Based on the description in the previous paragraph, the DEMO model of SMA can be constructed as follows. Because of space limitations, we focus here only on the organization construction diagram (OCD) of the construction model (CM). The unit of business service of SMA is the delivery of an IT solution to the customer. Therefore, the first transaction to be identified is the (T1) project completion. The customer (CA1) initiates this transaction by requesting an SMA employee to provide an IT solution. This employee will be called the project completer (A1). A1 initiates four transactions: project fee payment (T2), project planning (T3), project monitoring (T5) and task completion (T4). Note that T4 cannot be requested before the project plan is complete. Because the project manager controls the execution of the plan, he or she takes the role of task manager (A5). This actor initiates and periodically executes (every week) the transaction project monitoring (T5) tasks. Execution of this transaction leads to changes in the project plan. To model the salary and rewards mentioned in the description, salary payment (T6) and reward payment (T8) transactions are needed. To execute T6, we need to calculate the salary. Because there is no original fact in T7 (only calculation), it is an infological transaction (green). To execute T8, we need two sub-transactions: reward decision making (T9) and employee evaluation (T10). To perform the evaluation, another calculation transaction is needed (T11). The OCD of the CM is shown in Figure 6-2.

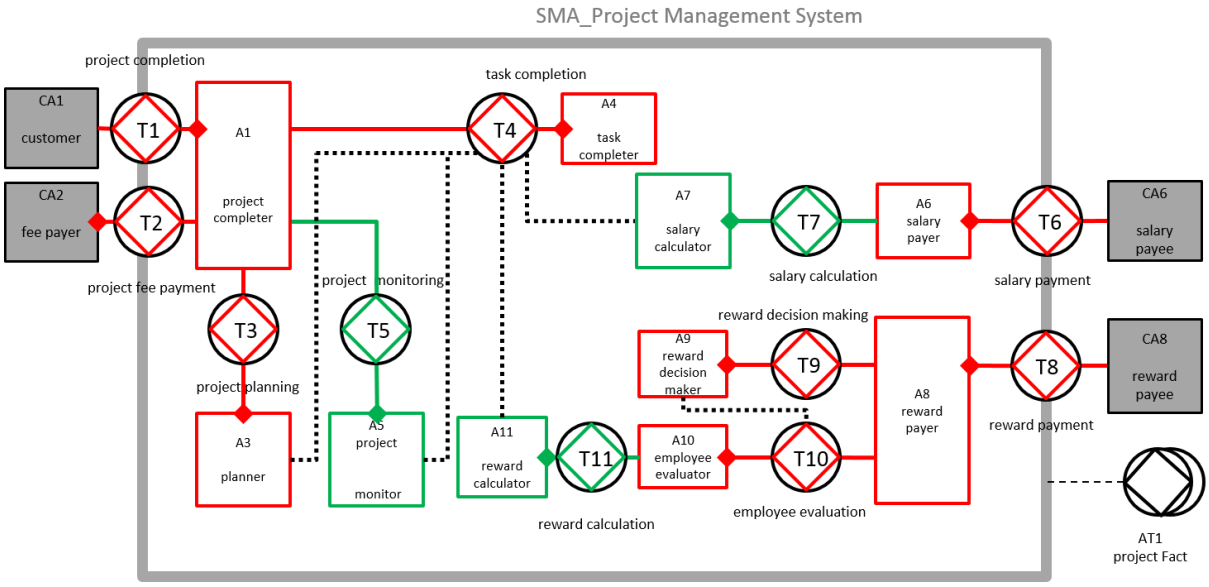


Figure 6-2: OCD for SMA

The CM consists of OCD and TPT. Table 6-1 shows the TPT for the SMA DEMO model.

Table 6-1: TPT for SMA DEMO

Transaction	Product
T1 project completion	P1 the project is complete
T2 project fee payment	P2 the project fees have been paid
T3 project planning	P3 the project plan has been created
T4 task completion	P4 task is complete
T5 project monitoring	P5 monitoring of Project for Period is complete
T6 salary payment	P6 salary of Employee for Period has been paid
T7 salary calculation	P7 salary of Employee for Period has been calculated
T9 reward decision-making	P9 the reward decision of Employee for Period has been made
T8 reward payment	P8 the reward of Employee for Period has been paid
T10 employee evaluation	P10 the evaluation of Reward of Employee for Period has been made
T11 reward calculation	P11 the reward of Employee for Period has been calculated

As we can see from the OCD, there are three main parts: project completion, salary, and reward. In the following diagrams, we will illustrate the Process Structure Diagram (PSD) of the PM. This diagram shows the sequences during execution of the transactions to determine the reward for each employee.

Figure 6-3 shows the PSD of project completion part of SMA DEMO model. Project fee payment, project planning, task completion and project monitoring transactions are initiated after promising the project completion transaction. And they must be finished before the execution of the project completion. Before starting task completion transaction, project planning transaction must be finished. It means that the plan must be ready before starting to execute the tasks of the project.

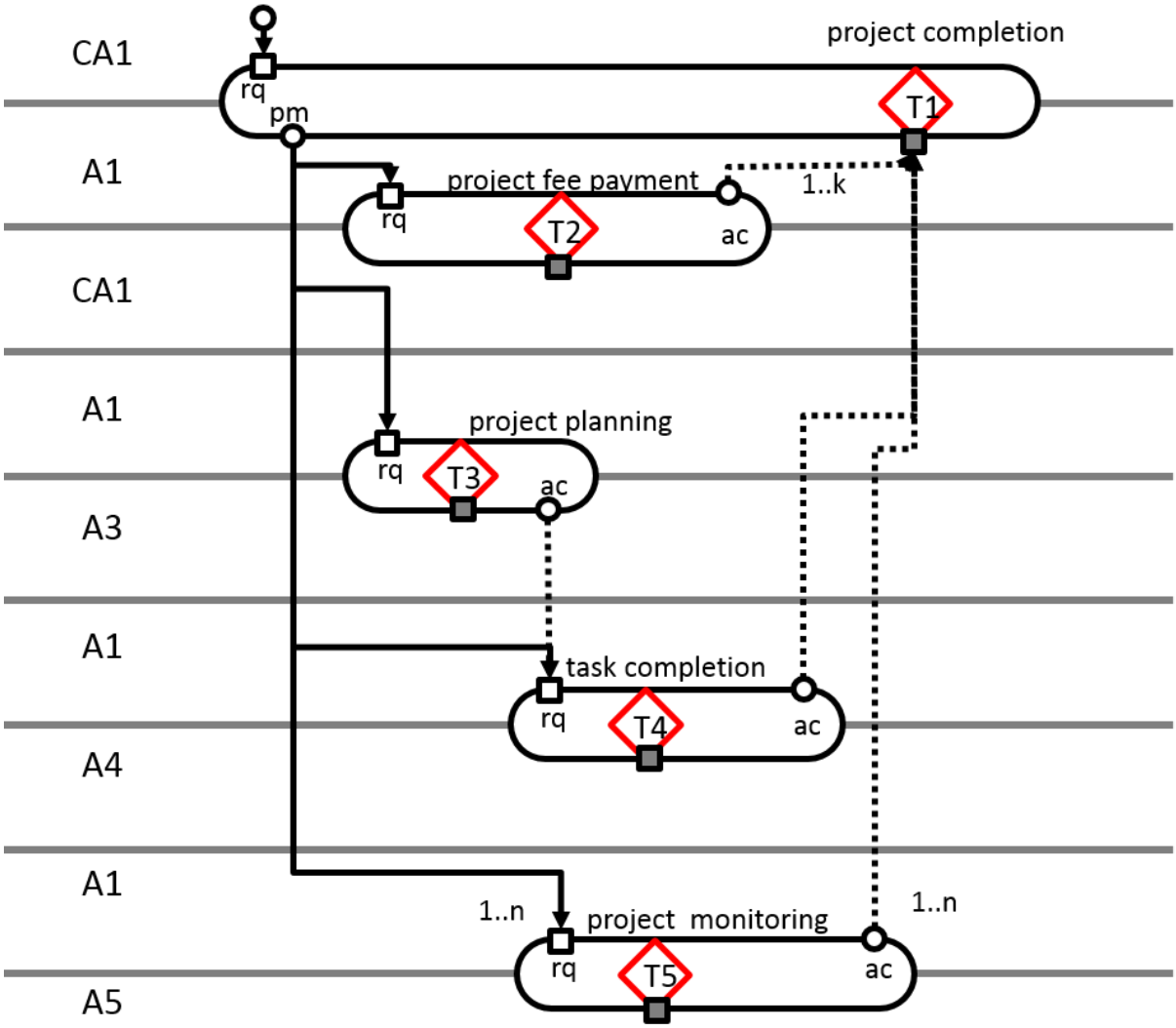


Figure 6-3:A project completion part of the PSD for SMA

Figure 6-4 shows the PSD of the salary part of SMA DEMO model. It consists of two transactions, salary payment, and salary calculation. Salary calculation is green because it is infological transaction.

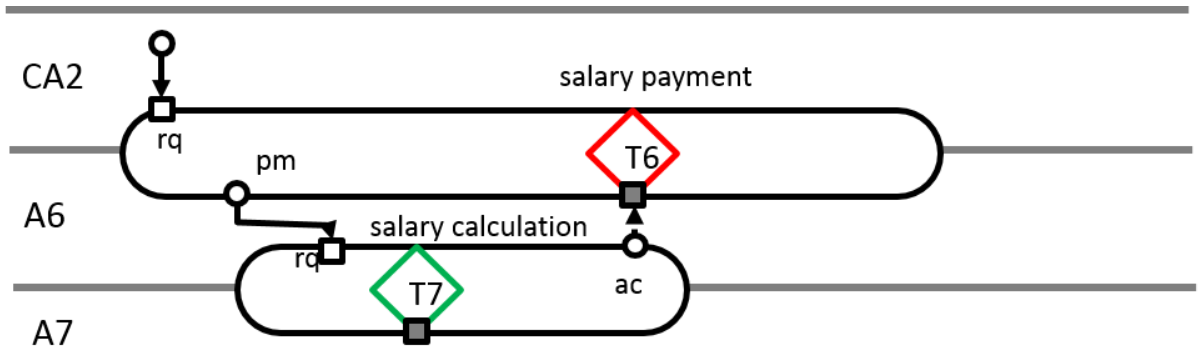


Figure 6-4: Salary part of PSD of SMA DEMO model

Figure 6-5 shows the PSD of the reward system of SMA. The reward payment transaction is initiated periodically. Therefore, it is a self-initiating transaction. From this diagram, a reward system based on three steps first is calculating all the numerical key performance indicators KPI. Second is evaluating the performance based on the result of the previous calculation. This transaction is red (ontological) because it the result is not automatic rather than the project manager should evaluate the performance of the employee. Third is making the decision of the amount of the reward that should be paid to the employee. This step is the based not only on the result of the employee performance but also on the budget and the cost and the total result of the project.

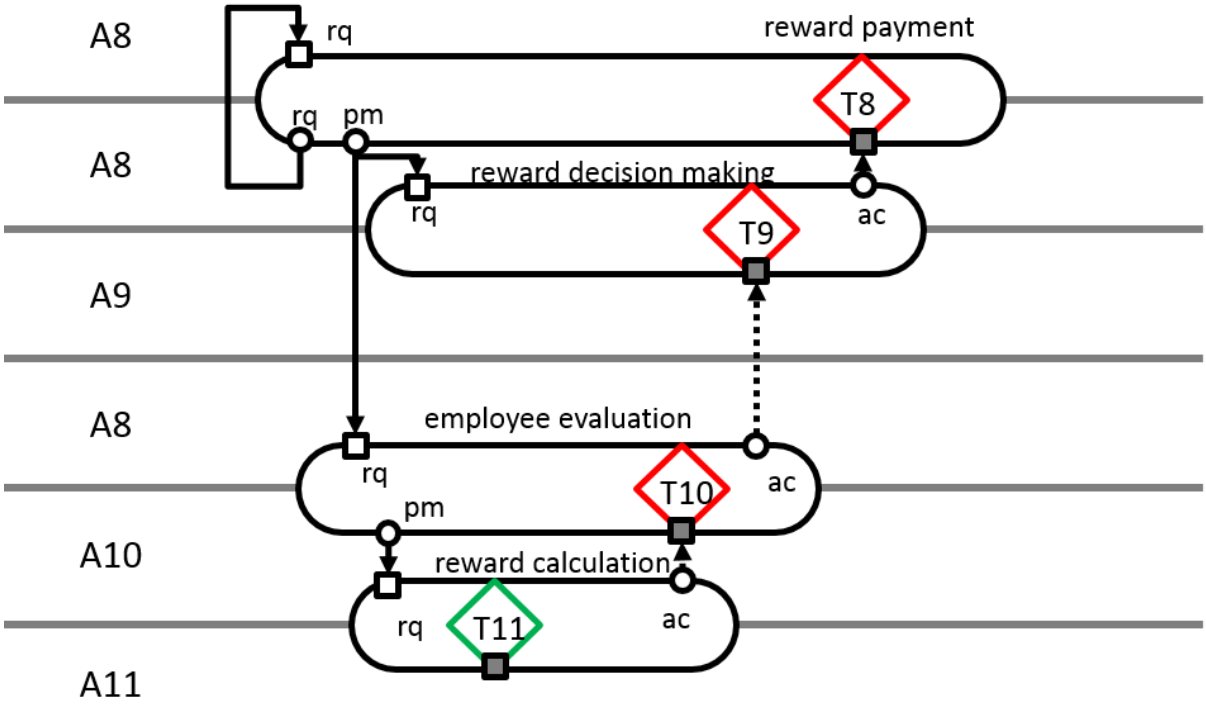


Figure 6-5: reward part of PSD of SMA DEMO model

The third model of DEMO is Fact Model. The diagram of FM is represented in Object Fact Diagram (OFD). Figure 6-6 shows the first part of OFD. OFD shows the objects which are involved in the enterprise and the relationships between them by the facts. From figure 6-6, Project object is the main object in this diagram.

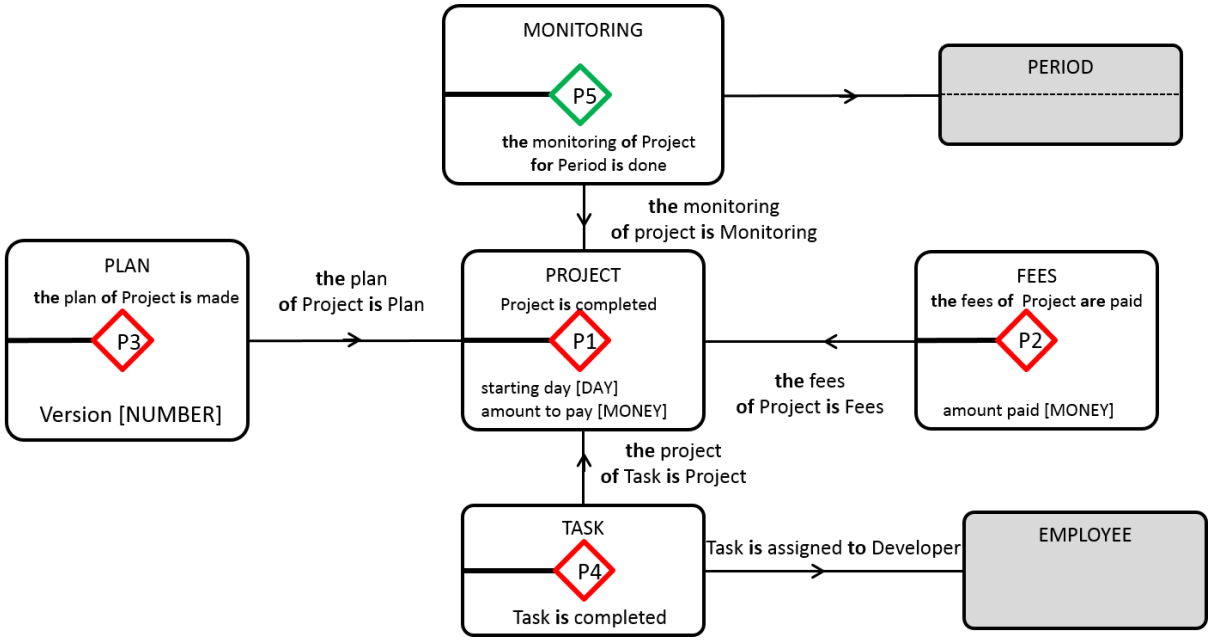


Figure 6-6:A project completion part of the OFD for SMA

Figure 6-7 shows the OFD part of salary payment.

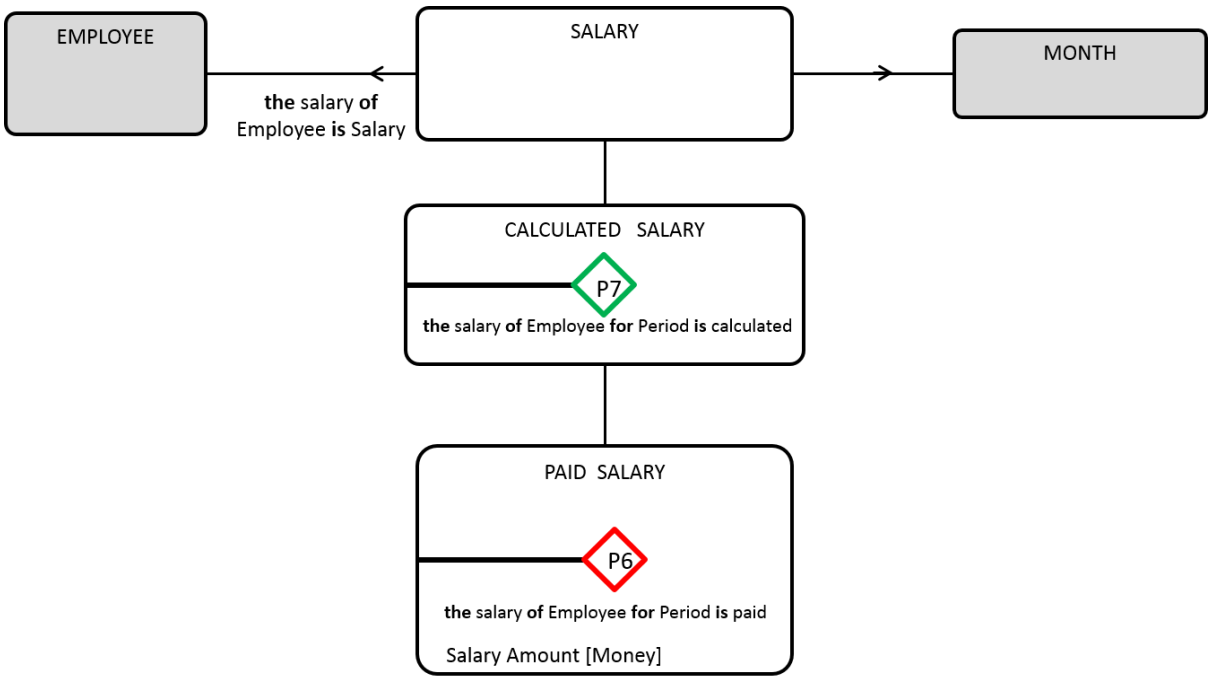


Figure 6-7: Salary part of OFD of SMA DEMO model

Figure 6-8 shows the OFD for the reward part. It is clear that the reward object is the main one in this diagram.

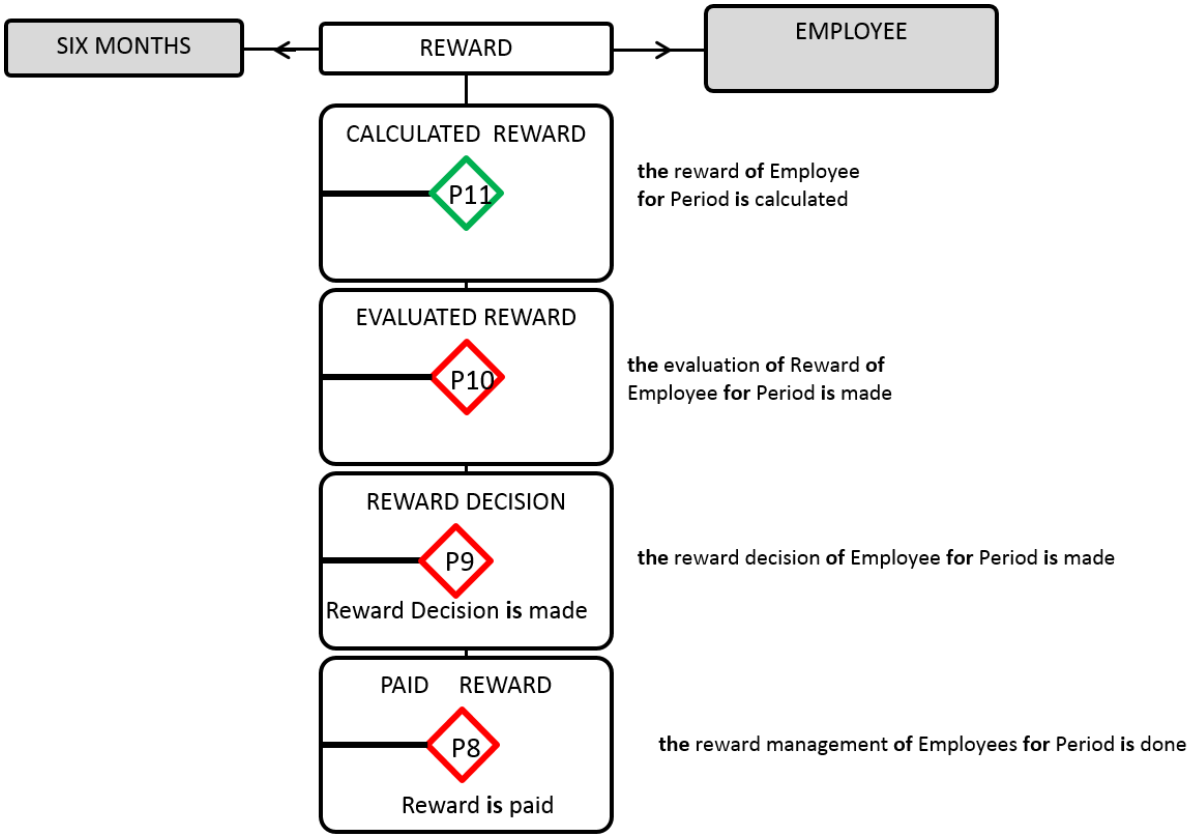


Figure 6-8: reward part of OFD of SMA DEMO model

The last DEMO model is the Action Model. For each actor, there are several process steps like request and promise. For each one there is action rules specification. Sometimes it is possible to combine more one step in one action rule specification. Figure 6-9 shows the action rule specification for the Actor1 in the process step request (rq).

when new Project completion **is** requested
(T1/rq)
with **the** start date **of** Project = <some Day>
the end date **of** Project = <some Day>
the priority **of** Project) **is** <some Priority>
the customer **of** Project **is** <some Customer>
the title **of** Project **is** <some Title>
the description **of** Project **is** <some Description>
if
check justness: performer of [act] **is** authorized **and**
the business context is valid **and**
check sincerity: performer of [act] **is** sincere
and
check truthfulness: *there is no objection against proceeding*
then project completion **of** Project **must be** promised
[T1/pm]

Figure 6-9: Action Rule Specifications for A1 (1)

when project completion **of** Project **is** promised
(T1/pm)
if
check justness: performer of [act] **is** authorized **and**
the business context is valid **and**
check sincerity: performer of [act] **is** sincere
and
check truthfulness: *there is no objection against proceeding*
then Project completion **must be** produced
[T1/ex]
the fees **of** Project **is** paid **must be** requested
[T2/rq]
the plan **of** Project **is** made **must be** requested
[T3/rq]
Task **is** completed **must be** requested
[T4/rq]
the monitoring **of** Project **for** Period **is** done **must be** requested
[T5/rq]
Project completion **must be** stated
[T1/st]

Figure 6-10Action Rule Specifications for A1 (2)

when the fees **of** Project **is** paid **is** stated
(T2/st)
with **the** amount = <money>
if
check justness: performer of [act] **is** authorized **and**
the business context is valid **and**
check sincerity: performer of [act] **is** sincere
and
check truthfulness: *there is no objection against proceeding*
then project fee payment **must be** accepted
[T2/ac]

Figure 6-11Action Rule Specifications for A1 (3)

For developing an IS for SMA and based on the framework the following steps are applied:

6.3 Developing IS for SMA:

STEP 1: SCOPING

IS Objectives:

The information system to be developed has three main objectives. The first objective is to follow up on the execution of all the projects. During project execution, the project manager needs to know the task statuses and the workload for each employee. This helps to ensure the project meets timeline, quality and cost limits. The second objective is automation. Because the salary of each employee is based on the tasks that are executed by that employee, several calculations are required. To reduce the cost of these calculations, they should be automated. The third objective is the performance analysis. To analyze the performance of each employee, a record of his/her achievements should be archived. Because each task is associated with an execution time, these data can be used to estimate the productivity of the employee.

IS Boundaries:

Based on the objectives of the IS, only related transactions are defined in the SMA OCD. This means that transactions related to marketing, recruiting and finance are already

excluded. Therefore, SMA OCD includes all the transactions related to the three objectives mentioned in the previous paragraph. However, during the analysis of the required IS with the stakeholders, it becomes apparent that the project fee payment and the project planning steps should not be embedded in the system. The project fee payment transaction occurs only rarely, so it should be paper based transaction without any relationship to the IS. In contrast, project planning is actually a very complex transaction. And because SMA has adopted the Agile methodology, it is very difficult to make a precise plan before beginning to execute the plan. Therefore, it is not required in the IS. The executor of T4 and the initiator of T6 and T8 were intentionally created as external actors. Those actor roles are fulfilled by employees who are represented as external composite actor roles for two reasons: First; to make it explicit that those employees serve the company through their work, which consists of an accumulation of tasks. In return, they are paid a basic salary and a reward to motivate them to improve their performance. Second, to avoid using the self-initiating transaction type for T6 and T8. Although T6 and T8 are periodically executed without an explicit request from the employee, it is important to show that they should accept the result of this transaction (salary and reward). This reflects their satisfaction just as if they were external customers, and that is why they are always called “internal customers.”

STEP 2: USERS ANALYSIS

Actor Role-Actor analysis:

After having several discussions with people involved in the project, we agreed on the following table, which represents the relationship between the actor roles and each actor.

Table 6-2: Relationships between Actors and Actor roles

DEMO actor role	Actor
customer, reward manager and reward decision maker	General Manager
project completer, project monitor and employee evaluator	Project Manager (Employee)
salary payee and reward payee	Employee
task completer (Employee)	Project Member
salary payer, salary calculator, reward	Accountant

Table 6-2 shows that the customer actor role is fulfilled by the general manager. This is because the real (external) customer has no direct link to the IS. After negotiating with the customer about the project, the general manager decides whether the company will start the project or not. From the table, we can see that the project manager, project member, and accountant are employees. This is useful for creating an inheritance relationship among the actors.

Transaction pattern:

After having several discussions and analyzing the requirements of the system, we found that the transaction patterns could not be implemented as initially designed. We had to adapt them to the specific needs of the users in SMA. The reason for the change is that some process steps in the transaction pattern are acted upon implicitly by direct discussion, and there is no need to store these communications in the IS. Therefore, we used the transaction pattern from DEMO solely as a reference to analyze the possible process steps required in the IS. Tables 5-7 show which pattern is used as a reference for each transaction. BTP denotes the basic

transaction pattern; STP denotes the standard transaction pattern, and CTP denotes the complete transaction pattern.

Table 6-3: Selecting Transaction patterns

Transaction	pattern
T1 project completion	BTP
T2 project fee payment	Out of scope
T3 project planning	Out of scope
T4 task completion	CTP
T5 project monitoring	BTP
T6 salary payment	STP
T7 salary calculation	BTP
T9 reward decision-making	BTP
T8 reward payment	STP
T10 employee evaluation	BTP
T11 reward calculation	BTP

As Table 6-3 shows, T4 task completion is the only transaction that uses the complete transaction pattern. That is because it is the main transaction in this enterprise—all the other transactions depend on its information. Additionally, the T4 transaction has the possibility of a promise or refusal or revocation. The other transactions can use BTP because only the happy path is applicable, and there is no reason to refuse the request or reject the result of the transaction.

Delegation

After having several discussions, we found the following delegations in the SMA DEMO models.

Table 6-4: Delegations in SMA DEMO

Transaction	Act	From actor role	To actor role
T4 task completion	Request	Project completer	Task completer
T1 project completion	Promise	Project completer	customer
T6 salary payment	request	Task completer	salary payee
T8 reward payment	request	Task completer	reward payee
T9 reward decision-making	Accept	Reward payee	Reward decision maker

From Table 6-4, the project completer may delegate the request to the task completer to let the employee initiate the task him or herself; however, the acceptance of the result of the task must still be performed by the project completer.

STEP 3: FUNCTION ANALYSIS

Use case diagram

After selecting the transactions, the IS will support, we defined required use cases for each transaction: 80 use cases in total. The following figure shows the derived use cases for the reward-related transactions.

Table 6-5: Use cases for reward-related transactions in SMA DEMO

Actor role	Actor	Use case	Transaction
reward payer	Accountant general manager	Pay reward	T8 reward payment
reward decision maker		decide reward	T9 reward decision-making
reward payer	accountant	calculate a reward for the user for a period	T11 reward calculation
reward payer	accountant General Manager	View rewards list for user	T8 reward payment
reward payer	accountant General Manager	View reward details	T8 reward payment
reward decision maker	Employee General manager	Edit reward decision	T9 reward decision-making
reward payer	accountant	Delete reward	T8 reward payment
reward payer	accountant	Edit reward details	T8 reward payment
Employee evaluator	Project manager	Evaluate employee performance for period	T10 employee evaluation
Employee evaluator	Project manager	Evaluate employee task performance	T10 employee evaluation
Employee evaluator	Project manager	View employee evaluation details	T10 employee evaluation

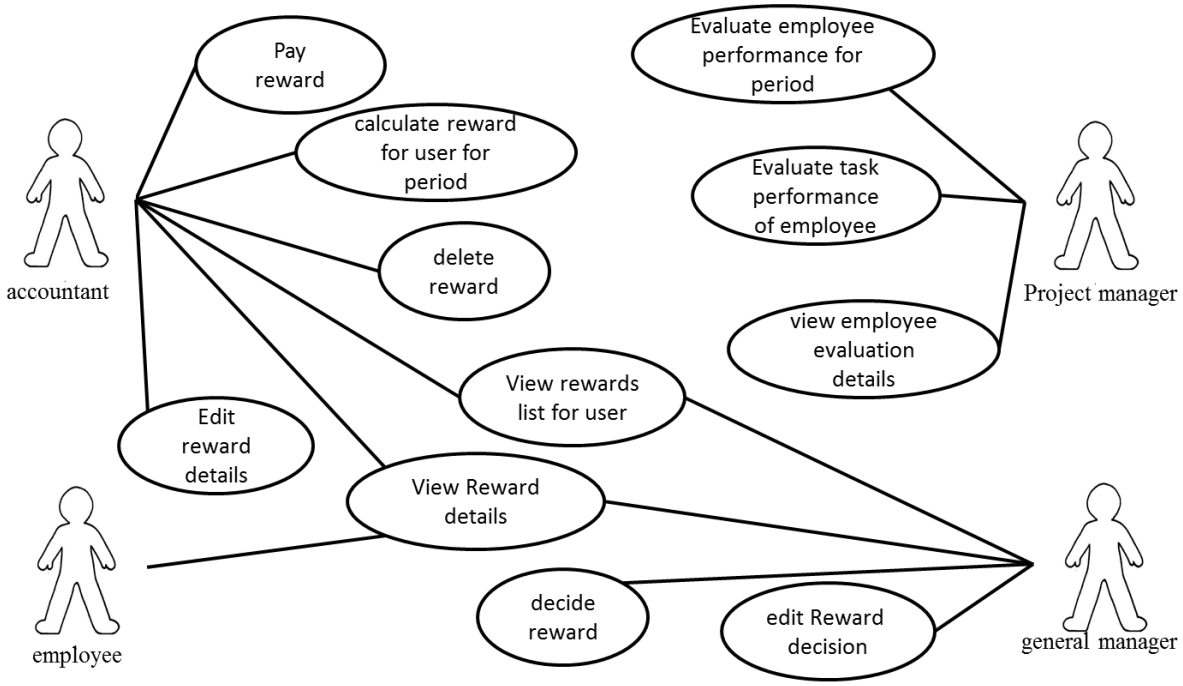


Figure 6-12: Use Case Diagram for Rewards

Use case scenarios

For each use case, we defined the scenarios. Table 6-5 shows an example of the Add New

Project use case.

Table 6-6: Add New Project Use Case

Use Case Name	**Add new project**
ID	**UC-01**
Description	This use case shows how a new project can be added to the system.
Actors	General Manager
Pre-condition	User is logged in. User has the permission of General Manager.
Main execution steps	Make request to add a new project Fill in the details of the new project Verify the details of the new project Agree on adding the new project with the filled in details
Alternative execution steps	If the project name is empty or a project already exists with the same name, then return to the new project details step. The user can cancel the Add New Project operation at any time.
Post-conditions	A new project is added

Activity diagram

When the use case scenario is complex, we may analyze it more closely by drawing an activity diagram to illustrate the activities involved in the use case.

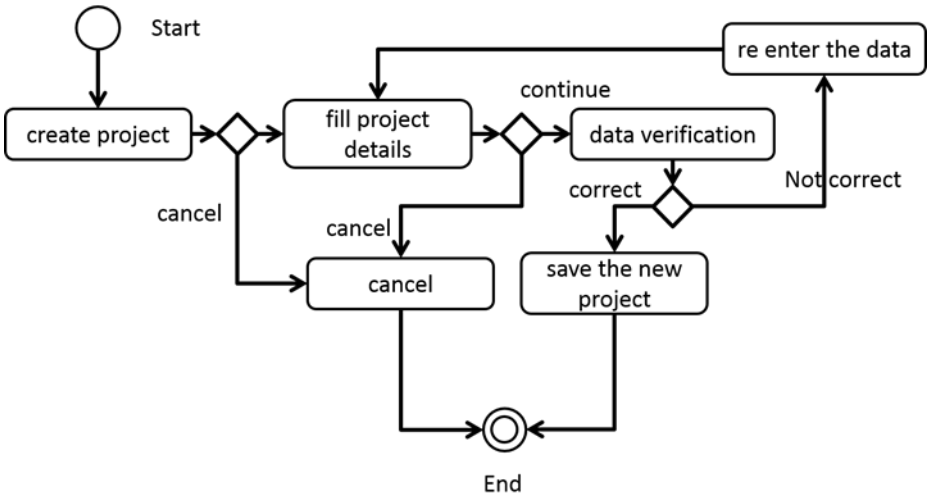


Figure 6-13: Activity diagram for use case Add New Project

STEP 4: DATA STRUCTURE:

Product Structure:

The structure of the products of the transaction can be made directly from the OCD. Figure 6-14 shows the product structure for SMA.

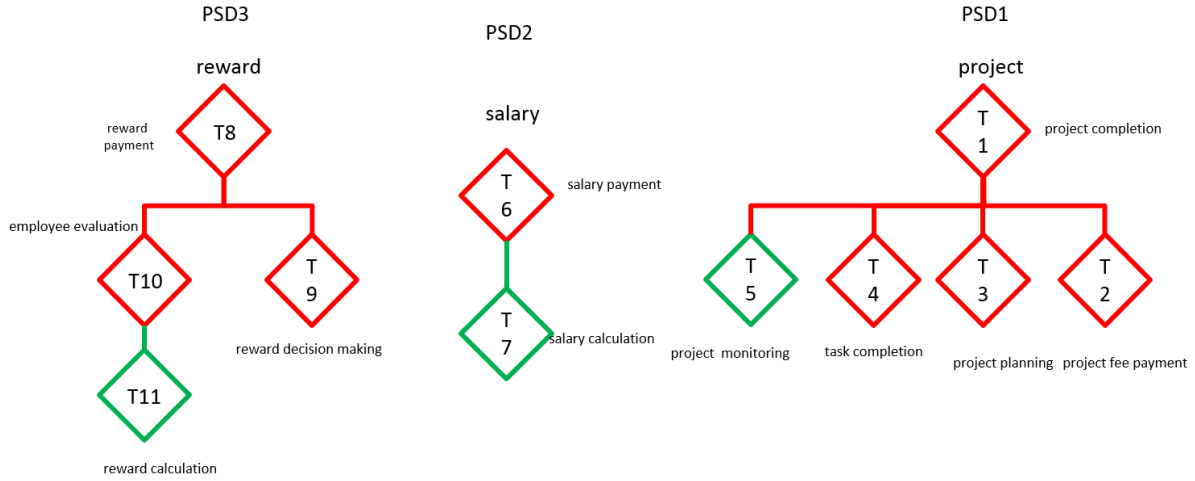


Figure 6-14: Product Structure of SMA DEMO model

From Figure 6-14, there are three main parts: reward, salary, and project. Those three parts are directly related to the objectives of the IS to be implemented.

Entity Relationship Diagram:

Deriving an ERD from the OFD of DEMO is an almost one-to-one operation. Each object from DEMO is replaced by an entity in ERD. Table 6-7 shows the objects from DEMO and their replacement entities in the ERD. From the table, note that PLAN and FEES were disregarded by the stockholders because they do not want the system to capture those two transactions in the IS. The MONITORING object is a product of a green transaction, meaning it is a derived fact, not an original fact. Therefore, there is no corresponding entity for it in the ERD. The MONTH, PERIOD and SIX MONTHS objects are time-related objects, which are transferred to the ERD as entity attributes. In the T4 task completion transaction, because we adopted the complete transaction pattern, we need additional entities to store the required information. Acts_Tasks stores all the acts related to a particular task. Similarly, Task_Delegations stores all the delegations for the initiating or executing actor role of a task. Task_Comments stores user comments on a task, providing a space for collaboration and discussion between the project team members. Acts_Lists is the set of all possible acts according to the complete transaction pattern. Revoke_Acts stores information about revoked acts. Finally, Task_Execution_Times stores the execution time for each task.

Table 6-7: Matching Objects of the FM from DEMO to Entities in the ERD

Object from DEMO	Entities in ERD	Comment
PROJECT	Projects	
PLAN	Out of scope	
TASK	Tasks Acts_Tasks Task_Delegations Task_Comments Acts_Lists Revoke_Acts Task_Execution_Times	
MONITORING	_	No original fact, only derived facts.
SALARY	Salaries	
FEES	Out of scope	
MONTH	_	Attribute, not entity
PERIOD	_	Attribute, not entity
SIX MONTHS	_	Attribute not entity
EMPLOYEE	Users Roles Roles_Users_Projects	OFD does not capture active entities
REWARD	Rewards	

Entity annotations:

Each term in the ERD must be clearly defined. Table 6-8 shows a glossary example for the reward entity attributes.

Table 6-8: Entities annotations of reward entity

Attribute	Type	Description
Id	Integer	Identifier defines each reward.
User_Id	Integer	The user who will receive the reward.
Amount	Integer	The amount of money to be paid as a reward.
Start_Period	datetime	Each reward is paid for a specific period of time. Start_Period defines the start time for the reward.
End_Period	datetime	Each reward is paid for a specific period of time. End_Period defines the end time for the reward.
Description	text	A description by the general manager is explaining the reason for this reward.

6.4 Findings

In this paragraph, a summary of the findings of the selected case study is presented.

Table 6-9: Results of the selected case study

Phase	DEMO	IS
Scoping	11 Transactions	9 Transaction
Users Analysis	13 Actor roles	5 Actors
Function Analysis	11 Transaction (6 red, 3 green)	82 use cases
Data Structure	7 red (not time-related) Objects	7 Entities
	3 red (time related) Objects	0 Entities
	None	7 Entities
	1 green	0 Entities

The previous table shows that two transactions were excluded in IS. It was easy for the decision maker to select which transaction to be included in IS. From the same table, five types of actors are fulfilling 13 actor roles. By using Actor-Actor role analysis, it was clear to define the responsibility for each employee in the company. From the selected 11 transactions, 82 use cases were derived. All the use cases derived based on the developed framework. As for the entities, for each one red object (not time-related) one entity is created. However, all the time related objects were modeled as attributes.

CHAPTER 7: CONCLUSION

7.1 Discussion

In this chapter, we will discuss the main benefits of our approach to developing information systems and the insights we obtained from applying this approach to one real-world case study. Based on the study, we found that DEMO contributes to information system development in many different ways, the most important of which are analyzing and specifying requirements. Because the DEMO model is concise and simple compared to many other modeling languages, it makes the system more understandable by the stakeholders involved in planning the IS. Having a good understanding of the enterprise backed by a solid model such as DEMO makes it easier to specify the requirements of the IS to be developed. In the DEMO model, the ontological transactions are clearly specified. Those transactions represent the functions that the enterprise executes to provide the environment with products and services. And because the actor roles are specified in the DEMO, their responsibilities are clear and available later in the development process. Therefore, there is no conflict in specifying the requirements. And if there are any errors or mistakes in the requirements, it is easy to ensure their correctness by validating them against the DEMO model of the enterprise. This has a great impact on aligning business and IT because DEMO models the enterprise business, and the IS is the way IT supports the business.

DEMO models the business of the enterprise as well as—in an abstract way—the implementation. And the IS supports the business of the enterprise. Therefore, DEMO plays an important role in both creating the roadmap of the IS and for planning how it might be modified in the future. We can consider the DEMO model as a solid documentation of the enterprise for any further development and maintenance of the IS.

Because DEMO captures all aspects of the enterprise within its four models, it contributes not only to requirements but also to the design of the IS, as shown earlier when we derived the ERD directly from the FM of DEMO. Similarly, the PM helps in modeling activity diagrams for the IS. Finally, the AM plays an important role in specifying the pre and post-conditions for each use case.

DEMO's ability to help with IS development increases when many users will interact through the IS. Such IS applications are called interactive information systems (Valente, 2009). DEMO models are based on modeling the transactions and the actor roles responsible for initiating and executing the transactions (interactivity); therefore, DEMO gains in usefulness as interactivity increases.

In previous research, the infological transaction and actor roles (green) were modeled just like the business ones (red). However, in our research, we found there is no need for that. In fact, considering the green transaction, the same as the red ones simply makes the model more complex. Green transactions are functions executed by the IS to deliver a particular value to users. In most cases, those functions merely manipulate or display already stored information in response to user needs.

Transaction patterns were used only as a reference when developing the IS. This is because many acts are performed implicitly by direct interaction between people in the enterprise; therefore, there is no need to explicitly store information about these interactions in the IS. In addition, transaction patterns do not consider collaborations between people; for example, some transactions may be initiated or executed collectively by many users, but that is not specified in the transaction patterns.

Finally, self-initiating transactions should be avoided when developing an IS. Transactions are clearer when there are always two actor roles—one responsible for initiating a transaction

and a different one responsible for executing the transaction—even though both roles are sometimes filled by one actor. The reason for this is to emphasize that, for each transaction, even those initiated only periodically, someone must be responsible for accepting or rejecting the product of that transaction. Another point we want to make is that the product of a transaction in the FM should not be attributed a time-related object but rather to the real object that includes it. For example, T6 salary payment is a transaction initiated by the employee and executed by the salary payer. The product of that transaction is "The salary of Employee for Period has been paid." Therefore, the product of this transaction belongs to the salary object, not the period object. The period object will be only an attribute of the salary entity in the ERD. If we instead consider that the product is "the salary for Period has been paid," it is misleading; salary is not a product that belongs to the Period object.

7.2 Conclusion and Future Work

To conclude this research, the following points clarify the important points of developing IS based on DEMO:

IMPROVING THE BUSINESS PROCESS BY SIMULATION BEFORE IS DEVELOPMENT

In the proposed approach, I run a simulation of DEMO model first. This plays an important role in understanding the functions of the enterprise and how they deliver it to the environment. If the model is not correct or need any correction, then stakeholders can notice that and point it out by this simulation. In the case study, DEMO model has been revised many times. In fact, 16 versions have been made. This highlights that any model of an enterprise need a deep discussion before we go to the next stage to develop any IS. It is very risky to start developing IS, and then discovering that the developed IS doesn't meet the needs. This is one the main problems that were discussed in the first chapter. And this problem motivated me to develop this approach.

DEFINING THE SCOPE OF IS BY DEMO MODELS

Defining the scope of IS is very difficult problem in the practice. If the scope is not clearly specified, then there will be no end of the project. In addition, the cost will exceed the budget. To avoid this problem and by using the approach in this research, stakeholders may have a meeting and agree on the DEMO model first. Then they may do another meeting to discuss and specify what are the transactions that need support by IS and how IS will support that transaction. Then the scope of IS could be defined by the transactions and the actors roles that must be included in the developed IS. In the case study, 11 transactions were defined. However, during the discussions with the stakeholders Transaction 2 and Transaction, 3 were excluded from the project of IS. The reason for excluding T2 is that it has direct communication with the customer. And SMA wants to use the IS internally. They don't want to force any of their customers to use their internal IS.

ANALYSING ACTOR ROLES IN IS

Defining the actor roles in any IS is a difficult task. This is because that many people in the enterprise may take more than one actor role. To respond to this problem in this research, all the actor roles within the enterprise were defined in DEMO model. Then with the discussion with the stakeholders, we could define the actor roles that every person in the enterprise is doing, and he or she should be responsible for. This is done in table 6-2. However, one important point is that people in the enterprise may change their actor roles based on their performance. Therefore, there must be a control panel in the developed IS allowing to change the actor roles for each person.

DEFINING THE FUNCTIONAL REQUIREMENTS OF IS

Having a good understanding of the enterprise backed by a solid model such as DEMO makes it easier to specify the requirements of the IS to be developed. In the DEMO model,

the ontological transactions are clearly specified. Those transactions represent the functions that the enterprise executes to provide the environment with products and services.

Because DEMO captures all aspects of the enterprise within its four models, it contributes not only to requirements but also to the design of the IS, as shown earlier when we derived the ERD directly from the FM of DEMO. Similarly, the PM helps in modeling activity diagrams for the IS. Finally, the AM plays an important role in specifying the pre and post-conditions for each use case.

DEFINING THE DATA STRUCTURE OF IS BASED ON FM OF DEMO

Data structure play an important role in developing IS. It is the skeleton of any information system. The database is defined based on this structure. And all the other elements of the software like the interface and the logic are based on it. If there is any error in the data structure, it needs longer time to fix it than if there is any error in the interface. To respond to this issue in this research, the data structure is directly derived from the FM of DEMO. As the case study showed that, every object from FM will be an entity except the time-related objects. Time-related objects would be attributes. However, Objects in FM reflects only the passive entities of the system excluding the users who use the system. Users are defined by the actor roles. Therefore, entities for users must be added to the resulted entity relationship diagram. Table 6-7.

DRAWBACKS OF THIS APPROACH

DEMO's ability to help with IS development increases when many users will interact through the IS. Such IS applications are called interactive information systems (Valente, 2009). DEMO models are based on modeling the transactions and the actor roles responsible for initiating and executing the transactions (interactivity); therefore, DEMO gains in usefulness

as interactivity increases. However, when the needed IS has a very few actor roles, then DEMO may not play an important role in defining the functions of IS.

To evaluate this research, I added a comparison with the previous research to clarify the advances of this research compared to the previous one figure 7-1. The first row the approaches of developing IS based on DEMO are listed. The first column represents the aspects of ISD to be compared. Circle means that the approach fully responds to that aspect. Trainable means it partially respond to that aspect. For example, in the previous research, only one use case is derived from each transaction 2005. However, in this research five main use cases are listed, as well all the possible needed use cases that are related to the whole system are listed. Moreover, the framework in deriving the entities is the first one to clarify the difference between time-related objects and the non-time related objects. And the other possible entities that are needed for the whole system are listed, too. This makes it easier to develop IS based on DEMO models.

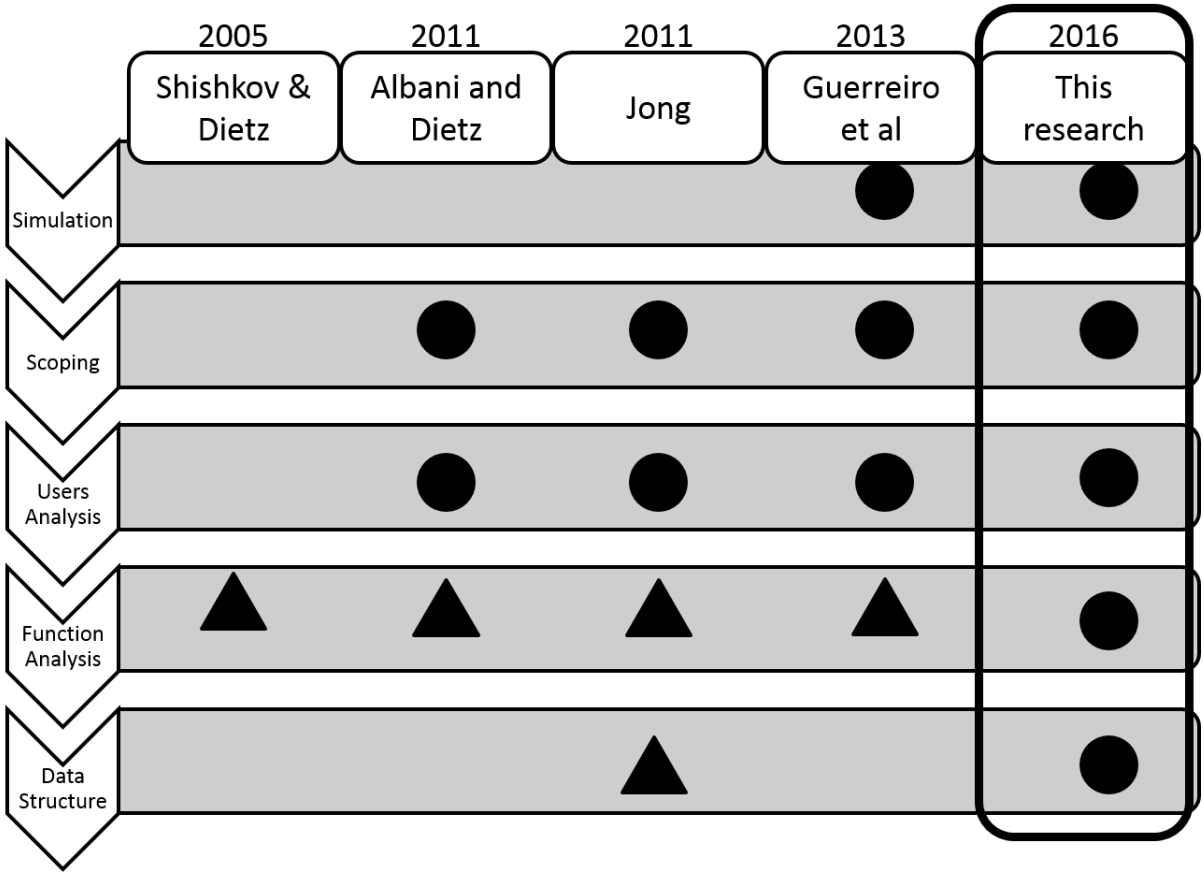


Figure 7-1 Comparison with Previous Approaches

In this research, I proposed a framework for developing ISs based on DEMO models. The framework derives IS artifacts from the DEMO model. This process is followed by a workflow that provides guidelines on how to implement the framework. The framework emphasizes analyzing the business transactions in the enterprise to derive the functional requirements of the planned IS. Moreover, the framework provides guidelines for deriving the entity relationship diagram from the Fact Model in DEMO. The framework was validated by a real-world case study. The results show that DEMO models can play an important role in analyzing and specifying requirements and in designing information systems. Moreover, to improve the quality of model a simulation methodology is proposed. This model simulation methodology can be considered a simulation based on a DEMO model, and this simulation can be used for any type of dynamic analysis. Some applications include the analysis and study of resource allocation problems, cost analysis, and the action time for each process.

This method could be used to optimize business processes. The analyses that can be conducted are structural and behavioral. There are many tools for conducting these analyses on Petri Net. All of them can be applied to analyze the proposed model. Using this analysis, deadlocks can be discovered, and exceptional cases handled, such as what if the transaction ends with a quit or stop.

Another contribution is that this model can be used to independently explain the DEMO Model. Despite the simplicity and conciseness of DEMO Models, it is difficult for most unfamiliar people to understand them, particularly for people who are used to addressing typical process models. By using this model, we can illustrate the DEMO Model using animations and examples, which allows for easy understanding of the important concepts of the DEMO Model. In fact, this model provides more insight into the enterprise and allows stakeholders to share interactively their ideas about the problem, which can lead to important discussions about the problem and how to solve it. Furthermore, it can be used to verify the constructed DEMO Model by executing many examples and showing them to experts.

The presented model is mainly based on the PSD of DEMO; however, we need to collect this information from the Action Model of DEMO to represent the conditions of the token movements. These conditions have been considered to be intuitive in this research. However, they are not in the formal transformation methodology. As a future project, the business rules that are expressed in the Action Model should be automatically addressed by this model.

One potential for transforming this model to a Petri Net Model is the possibility of taking advantages of existing Petri Net analysis tools and other tools to analyze the model.

An automatic transformation tool is needed to make it easy to perform this transformation.

Petri Net has a mathematical representation, which introduces many research possibilities for analyzing DEMO models mathematically. These possibilities can be studied in the future. As

for future work, there is a need to analyze the derivation of use case diagrams, activity diagrams, and ERDs to make the transformation more straightforward. Finally, developing tools to automate the transformation can help organizations that want to adopt this framework.

References

Aalst, W. M. (1995). A class of Petri nets for modeling and analyzing business processes. *Computing Science Reports 95/26, Eindhoven University of Technology*.

Aalst, W. M., van Hee, K. M., ter Hofstede, A. H., Sidorova, N., Verbeek, H. M., Voorhoeve, M., & Wynn, M. T. (2011). Soundness of Workflow Nets: Classification, Decidability, and Analysis. *Formal Aspects of Computing V23 (3)*, 333-363.

Albani, A., & Dietz, J. L. (2005). Identifying Business Components on the Basis of an Enterprise Ontology. *Proc. first Interop-ESA Conf. on Interoperability of Enterprise Software and Applications*, (pp. 359-370). Geneva.

Albani, A., & Dietz, J. L. (2006). The Benefit of Enterprise Ontology in Identifying Business Components. In A. D., E. S., K. J., & P.-H. J., *The Past and Future of Information Systems: 1976–2006 and Beyond* (pp. 243-254). Boston: Springer.

Albani, A., & Dietz, J. L. (2011). Enterprise Ontology Based Development of Information Systems. *Internet and Enterprise Management, 7(1)*, 41-63.

Albani, A., Keiblinger, A., Turowski, K., & Winnewisser, C. (2003). Domain Based Identification and Modelling of Business Component Applications. In K. e. L., *Advances in Databases and Information Systems* (pp. 30–45). Berlin: Heidelberg: Springer-Verlag.

Alencar, F., Silva, C., Moreira, A., Araújo, J., & Castro, J. (2006). Identifying Candidate Aspects with I-star Approach. *Proc. 5th International Conference on Aspect-Oriented Software Development*, (pp. 4-10). Bonn.

António, S., Araújo, J., & Silva, C. (2009). Adapting the i* Framework for Software Product Lines. In H. C.A., & P. G., *Advances in Conceptual Modeling - Challenging Perspectives* (pp. 286-295). Berlin: Heidelberg: Springer-Verlag.

Barjis, J. (2007). Automatic Business Process Analysis and Simulation based on DEMO. *Enterprise Information Systems*, 365-381.

Bera, P., & Evermann, J. (2014). Guidelines for Using UML Association Classes and Their Effect on Domain Understanding in Requirements Engineering. *Requirements Engineering , 19(1)*, 63-80.

Castro, J., Lucena, M., Silva, C., Alencar, F., Santos, E., & Pimentel, J. (2012). Changing attitudes towards the generation of architectural models. *Journal of Systems and Software*, 463–479.

DE JONG, J. (2011). Designing the Information Organization from Ontological Perspective. In A. A., D. J.L.G., & V. J., *Advances in Enterprise Engineering V ,79* (pp. 1-15). Berlin: Heidelberg: Springer-Verlag.

DE JONG, J. (2013). A Method for Enterprise Ontology based Design of Enterprise Information Systems. *doctoral diss., delft university of technology*. Mekelweg.

Dietz, J. L. (2003). Deriving Use Cases from Business Process Models. In S. Il-Y., L. S.W., L. T., & S. P., *Conceptual Modeling - ER* (pp. 131-143). Berlin: Heidelberg: Springer-Verlag.

Dietz, J. L. (2005). System Ontology and Its Role in Software Development. *Open Interop Workshop on Enterprise Modelling and Ontologies for Interoperability*. Rome: Springer.

Dietz, J. L. (2006). *Enterprise Ontology Theory and Methodology.* Springer-Verlag Berlin Heidelberg.

Dijkman, R. M., Dumas, M., & Ouyang, C. (2008). Formal Semantics and Analysis of BPMN Process Models using Petri Nets. *Information and Software Technology, Vol. 50, Issue 12*, 1281-1294.

Dumay, M., Dietz, J., & Mulder, H. (2005). Evaluation of DEMO and the Language/Action Perspective After 10 Years of Experience. *Proc. 10th Anniversary International Working Conf. on The Language-Action Perspective on Communication Modelling*, (pp. 77-105). Kiruna.

Entity-Relationship Diagram. (2009). In L. Q., & C. Y., *Modeling and Analysis of Enterprise and Information Systems* (pp. 125-139). Berlin: Heidelberg: Springer-Verlag.

Eshuis, R., & Wieringa, R. (2002). Comparing Petri Net and Activity Diagram Variants for Workflow Modelling - A Quest for Reactive Petri Nets. In *In Weber et al* (pp. 321--351). Springer.

Fatyani, T., Iijima, J., & Park, J. (2015). Comparing DEMO with i_Star In Identifying Software Functional Requirement. *Proc. of the 5th International symposium on Business Modeling and software Design*, (pp. 139-149). Milan.

Fatyani, T., Iijima, J., & Park, J. (2015). Enterprise Ontology-based Information Systems Development. *International Journal of Academic Scientific Research*, 76-100.

Fernández-Sáez, A. M., Generoa, M., Chaudronb, M. R., Caivanoc, D., & Ramosd, I. (2015). Are Forward Designed or Reverse-Engineered UML Diagrams More Helpful for Code Maintenance?: A Family of Experiments. *Information and Software Technology*, *57*, 644–663.

Figueiredo, M. C., Souza, C. R., Pereira, M. Z., Prikladnicki, R., & Audy, J. L. (2014). Knowledge transfer, translation and transformation in the work of information technology architects. *Information and Software Technology*, 1233–1252.

Fitsilis, P., Gerogiannis, V., & Anthopoulos, L. (2014). Role of Unified Modelling Language in Software Development in Greece – Results from an Exploratory Study. *ET Software , 8(4)*, 143 -153.

Garousi, G., Garousi-Yusifoğlu, V., Ruhe, G., Zhi, J., Moussavi, M., & Smith, B. (2015). Usage and Usefulness of Technical Software Documentation : an Industrial Case Study. *Information and Software Technology, 57*, 664–682.

Girault, C., & Valk, R. (2003). *Petri Nets for Systems Engineering.* Springer-Verlag Berlin Heidelberg.

Gorschek, T., Tempero, E., & Angelis, L. (2014). On The Use of Software Design Models in Software Development Practice. *Journal of Systems and Software*, 176–193.

Guerreiro, S., Kervel, S. v., & Babkin, E. (2013). Towards Devising an Architectural Framework for Enterprise Operating Systems. *Proc. 8th International Conference on Software Paradigm Trends*, (pp. 578-586). Reykjavik.

Heiner, M., Liu, F., Rohr , C., & Schwarick, M. (2012). Snoopy – a unifying Petri net tool. *In Proc. PETRI NETS* (pp. 398–407). Hamburg, Springer, LNCS.

Jensen, K., & Kristensen, L. M. (2009). *Coloured Petri Nets.* Springer-Verlag Berlin Heidelberg.

Kervel, S. J., Dietz, J. L., John, H., Meeuwen, T. v., & Zijlstra, B. (2012). Ontology driven enterprise information systems engineering. *7th International Conference on Software Paradigm Trends*, (pp. 205-210). Rome.

Kervel, S. v., John , H., Meeuwen, T. v., Vermolen, J., & Zijlstra, B. (2011). A Professional Case Management System in Production, Modeled and Implemented using DEMO. *In: 30th International Conference CBI2015*, (pp. 62-77). Brussels.

Krouwel, M. R., & Op 't Land, M. (2011). Combining DEMO and Normalized Systems for Developing Agile Enterprise Information Systems. *Proc. first Enterprise Engineering Working Conf.*, (pp. 31-45). Antwerp.

Letsholo, K. J., Chioasca, E.-V., & Zhao, L. (2013). An Integrative Approach to Support Multi-Perspective Business Process Modeling. *International Journal of Services Computing*, 11-24.

Liu, L., Yang, C., Wang, J., Ye, X., Liu, Y., Yang, H., & Liu, X. (2014). Requirements model driven adaption and evolution of Internetware. *Science China Information Sciences* , 1-19.

Mallens, P., Dietz, J., & Hommes, B.-J. (2001). The Value of Business Process Modeling with DEMO Prior to Information Systems Modeling with UML. *Proc. 6th CAiSE/IFIP8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design*, (pp. 31-47). Utrecht.

Malta, Á., Soares, M., Santos, E., Paes, J., Alencar, A., & Castro, J. (2011). iStarTool: Modeling Requirements Using the i* Framework. *Proc. of the 5th International i* Workshop*, (pp. 163-165). Trento.

Marwan, W., Rohr, C., & Heiner, M. (2012). Petri Nets in Snoopy: A Unifying Framework for the Graphical Display, Computational Modelling, and Simulation of Bacterial Regulatory Networks. In *Humana Press, Methods in Molecular Biology, Chapter 21 - preprint* (pp. 1-31).

Mazón, J.-N., Pardillo, J., & Trujillo, J. (2007). A Model-Driven Goal-Oriented Requirement Engineering Approach for Data Warehouses. *Proc. ER 2007 Workshops CMLSA, FP-UML, ONISW, QoIS, RIGiM,SeCoGIS*, (pp. 255-264). Auckland.

Michalik, B., Keutel, M., & Mellis, W. (2014). Coping with Requirements Uncertainty -- A Case Study of an Enterprise-Wide Record Management System. *System Sciences (HICSS), 2014 47th Hawaii International Conference on* (pp. 4024-4033). Waikoloa, HI: IEEE.

Motameni, H., Movaghar, A., & Fadavi Amiri, M. (2007). Mapping Activity Diagram to Petri Net. *International Journal of Engineering*, 65-76.

Mylopoulos, J., Yu, E., Giorgini, P., & Maiden, N. (2011). *Social Modeling for Requirements Engineering.* London: The MIT Press.

Op ’t Land, M., & Dietz, J. L. (2012). Benefits of Enterprise Ontology in Governing Complex Enterprise Transformations. *Proc. Second Enterprise Engineering Working Conf.*, (pp. 77-92). Delft.

Oriol, M., Marco, J., & Franch, X. (2014). Quality models for web services: A systematic mapping. *Information and Software Technology, vol. 56, no. 10*, 1167-1182.

Pandey, D., Suman, U., & Ramani, A. (2010). An Effective Requirement Engineering Process Model for Software Development and Requirements Management. *Advances in Recent Technologies in Communication and Computing (ARTCom), 2010 International Conference on* (pp. 287 - 291). Kottayam: IEEE.

Paul, D., & Tan, Y. L. (2015). An Investigation of the Role of Business Analysts in IS Development. *ECIS 2015 Completed Research Papers*, (p. paper 142). Münster.

Petre, M. (2014). “No shit” or “Oh, shit!”: Responses to Observations on the Use of UML in Professional Practice. *Software & Systems Modeling , 13(4),* 1225-1235.

Reinhartz-Berger, I., & Sturm, A. (2012). Comprehensibility of UML-based Software Product Line Specifications. *Empirical Software Engineering ,19(3),* 678-713.

Scanniello, G., Gravino, C., Genero, M., Cruz-Lemus, J. A., & Tortora, G. (2014). On the Impact of UML Analysis Models on Source-Code Comprehensibility and Modifiability. *ACM Transactions on Software Engineering and Methodology , 23(12,13),* 1-26.

Shishkov, B., & Dietz , J. L. (2001). Analysis of Suitability, Appropriateness and Adequacy of Use Cases Combined with Activity Diagram for Business Systems Modeling. *Proc. Third International Conf. on Enterprise Information Systems,* (pp. 854-858). Setúbal.

Shishkov, B., & Dietz, J. L. (2004). Deriving Use Cases from Business Processes. In C. e. O., *Enterprise Information Systems V ,3* (pp. 249-257). Netherlands: Springer Science + Business Media.

Shishkov, B., & Dietz, J. L. (2004). Design of Software Applications Using Generic Business Components. *Proc. 37th Hawaii Conf. on System Sciences,* (pp. 1-10). Big Island.

Shishkov, B., & Dietz, J. L. (2005). Applying Component-Based UML-Driven Conceptual Modeling in SDBC. *Proc. 21st International Conf. on Electrical Communications and Computers,* (pp. 82 - 87). San Andres.

Silva, I. F., Neto, P. A., O’Leary, P., Almeida, E. S., & Meira, S. R. (2014). Software Product Line Scoping and Requirements Engineering in a Small and Medium-Sized Enterprise: An industrial case study. *Journal of Systems and Software,* 189–206.

Staines, T. (2008). Intuitive Mapping of UML 2 Activity Diagrams into Fundamental Modeling Concept Petri Net Diagrams and Colored Petri Nets. *Engineering of Computer Based Systems, 2008. ECBS 2008. 15th Annual IEEE International Conference and Workshop on the* (pp. 191 - 200). Belfast: IEEE.

Terlouw, L. I., & Albani, A. (2013). An Enterprise Ontology-Based Approach to Service Specification. *Services Computing, IEEE Transactions on, 6(1)*, 89-101.

Tuunanen, T., Rossi, M., Saarinen, T., & and Mathiassen, L. (2007). A Contingency Model for Requirements Development. *Journal of the Association for Information Systems*.

Valente, P. (2009). *Goals Software Construction Process.* VDM Verlag Dr. Müller.

Weske, M. (2012). *Business Process Management,Concepts, Languages, Architectures.* Springer-Verlag Berlin Heidelberg.

Williams, B., & Carver, J. (2014). Examination of the Software Architecture Change Characterization Scheme Using Three Empirical Studies. *Empirical Software Engineering , 19(3)*, 419-464.

Wright, K., & Capps, C. (2010). A Survey of Information Systems Development Project performance. *Academy of Information & Management Sciences*, 87-105.

Appendix 1 Glossary

Term	Definition
Act	The atomic unit of action in an organisation. Two kinds of acts are distinguished: coordination acts and production acts. Acts are performed by actors
Action Model (AM)	One of the four sub-models of the ontological model of an aspect organisation. It consists of a set of action rules
action rule	A procedural guideline for dealing with an agendum kind. It consists of three consecutive parts: the event part, the assess part, and the response part
Activity Diagram (AD)	Activity diagram is basically a flow chart to represent the flow form one activity to another activity. The activity can be described as an operation of the system
actor role	the unit of authority, responsibility, and competence in organisations
Attribute type	a property type that is a purely mathematical mapping from an object class or a value scale to a value scale
Basic transaction pattern	In DEMO, There are three patterns of the transaction execution process. BTP has only the basic process steps (request, promise, state and accept)
B-organisation	(B from business) the aspect organisation of an enterprise that brings about the business of the enterprise. It consists primarily of original actors and original transactions, but it may also include informational and documental actors and transactions

Business	term to refer to the function perspective on an enterprise, in particular the function of the enterprise as perceived by the consumers of its services
Business process model	A tree structure of transaction kinds (possibly only one) in the B organisation of an enterprise. Every transaction kind in the tree is enclosed in a transaction kind on the next higher level (except the root transaction kind), and encloses one or more transaction kinds on the next lower level
Business Process Model and Notation (BPMN)	Business Process Model and Notation (BPMN) is a graphical representation for specifying business processes in a business process model
causal link	A link in a Process Model between a coordination act and a coordination fact, indicating that the fact is the result of the act. (Note: in the 'contracted' version of the Transaction Process Diagram, the causal links are 'hidden' in the combined symbol for the coordination act and the resulting coordination fact).
class diagrams	In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.
Coloured Petri Net (CPN)	Coloured Petri nets (CPN) are a backward compatible extension of the concept of Petri nets. CPN preserve useful properties of Petri nets and

	at the same time extend initial formalism to allow the distinction between tokens.
complete transaction pattern (CTP)	In DEMO, There are three patterns of the transaction execution process. CTP includes all the process steps and in the STP and the revocation process steps, too.
Composite actor role (CA)	a composite actor role consists of two or more elementary actor roles and the transaction kinds between them
Construction Model	One of the four sub-models of the ontological model of an aspect organisation .It contains the identified transaction kinds and actor roles, as well as the initiator links, the executor links, and the information links between them.
coordination acts (C-acts)	The atomic act in transactions. The result of a successfully performed coordination act is the creation of the corresponding coordination fact
Coordination fact (C – fact)	An elementary state of affairs in the coordination world. A coordination fact is the result of a successfully performed coordination act
Datalogy (blue)	A datalogical production act is an act in which one manipulates the form of information, commonly referred to as data, so without being concerned about its content. In DEMO, it is represented by blue color.
dependum	The dependum can be a soft-goal, a goal, a task or a resource. Soft-goals.
Design and Engineering	Design & Engineering Methodology for Organizations (DEMO) is an enterprise modelling methodology for transaction modelling, and

Methodology for Organizations (DEMO)	analysing and representing business processes. It is developed since the 1980s by Jan Dietz and others, and is inspired by the language/action perspective.
Edge Expressions	In Petri net, Edge Expressions is a formula that represents a condition to the token to pass from the previous place to the next place.
enterprise engineering	The engineering discipline that has enterprises as its objects of interest. Enterprise engineering particularly concerns the (re-) design (function design, construction design, and implementation design) of enterprises, thereby striving for the achievement of three generic goals: intellectual manageability, organisational concinnity, and social devotion.
Enterprise Information System	Business software application that serves the enterprise.
Enterprise Ontology	Conceptually, enterprise ontology is the implementation independent understanding of an enterprise's organisation. Practically, it is the PSI-theory based model of an enterprise's organisation. Enterprise ontology contributes in particular to achieving intellectual manageability.
Enterprise operating system	It is a system includes the DEMO engine with the running DEMO model of the enterprise.
Entity Relationship	In software engineering, an entity–relationship model (ER model) is a data model for describing the data or information aspects of a

Diagram (ERD)	business domain or its process requirements, in an abstract way that lends itself to ultimately being implemented in a database such as a relational database.
Executor	A role of an actor (role) in a transaction (kind). An actor (role) who is authorised to be the executor (role) of a transaction (kind), is responsible for performing the production act and the corresponding coordination acts, according to the transaction pattern.
executor link	a link in a Construction Model between an actor role and a transaction kind, indicating that the actor role is the executor (role) of the transaction kind.
Fact Model (FM)	One of the four sub-models of the ontological model of an aspect organisation. It contains the identified business object classes, business fact types, and business laws in the production world of the aspect organisation.
functional requirements	In software engineering (and systems engineering), a functional requirement defines a function of a system and its components. A function is described as a set of inputs, the behavior, and outputs.
Guard function	It maps each transition $t \in T$ into guard expression g. The output of the guard expression should evaluate to Boolean value true or false.
I star (i*)	i* (pronounced "I star") or i* framework is a modeling language suitable for an early phase of system modeling in order to understand the problem domain.
Infology (green)	A infological production act is an act in which one manipulates the

	original artefact by transforming it from one shape to another by some processing like calculation. In DEMO, it is represented by green color.
Information	the expression of (some) thought in (some) transmittable form.
Information Systems (IS)	An information system is any organized system for the collection, organization, storage and communication of information.
Initiator	a role of an actor (role) in a transaction (kind). An actor (role) who is authorised to be the initiator (role) of a transaction (kind), is responsible for performing the corresponding coordination acts, according to the transaction pattern.
Integration DEFinition (IDEF)	IDEF, initially abbreviation of ICAM Definition, renamed in 1999 as Integration DEFinition,[2] refers to a family of modeling languages in the field of systems and software engineering. They cover a wide range of uses, from functional modeling to data, simulation, object-oriented analysis/design and knowledge acquisition.
Model Driven Architecture (MDA)	Model-driven architecture (MDA) is a software design approach for the development of software systems. It provides a set of guidelines for the structuring of specifications, which are expressed as models. Model-driven architecture is a kind of domain engineering, and supports model-driven engineering of software systems.
Non functional requirements (NFR)	In systems engineering and requirements engineering, a non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific

	behaviors. They are contrasted with functional requirements that define specific behavior or functions.
Normalized Systems (NS)	Normalized Systems is a theory to design and engineer information systems exhibiting proven evolvability.
Object Fact Diagram	representation form of the Fact Model.
Ontology (red)	A ontological production act is an act in which one creates a new artifact. In DEMO, it is represented by red color.
Organisation Construction Diagram (OCD)	representation form of the Construction Model.
Petri Net (PN)	A Petri net (also known as a place/transition net or P/T net) is one of several mathematical modeling languages for the description of distributed systems.
Place	In Petri Net, place represents a state.
Process Model (PM)	one of the four sub-models of the ontological model of an aspect organisation. It contains the identified transaction kinds and the specific way in which they are interrelated through response links and waiting links.
Process Structure Diagram	representation form of the Process Model.
Product Structure	A tree structure of how the production facts are related to each other.
production acts	the act in a transaction by which the executor brings about the

(P-acts)	product.
Production fact (P – fact)	an elementary state of affairs in the production world. Production facts are the results of successfully carried out transactions. A distinction is made between (existentially) independent and dependent production facts. Independent production facts, also called products, are the direct results of transactions. There are two sorts of dependent production facts: properties and attributes.
Standard Transaction Pattern	In DEMO, There are three patterns of the transaction execution process. STP has all the process steps in BTP and the (decline, reject, stop and quit).
strategic dependency model (SD)	Set of nodes and links where each node represents an actor and each link between two actors indicates that one actor depends on the other for something in order that the former may attain some goal.
strategic rational model (SR)	The SR model is a graph, with several types of nodes and links that work together to provide a representational structure for expressing the rationales behind dependencies.
Transaction	the molecular unit of action in an organisation, consisting of a number of acts, according to the complete transaction pattern. The result of a successfully performed transaction is the creation of a product.
Transaction product table	table of transaction kinds and their corresponding product kinds.
Transition	In Petri Net, transition represents a transformation to an elements (token).

Unified Modeling Language (UML)	The Unified Modeling Language (UML) is a general-purpose, developmental, modeling language in the field of software engineering, that is intended to provide a standard way to visualize the design of a system.
use cases	In software and systems engineering, a use case is a list of action or event steps, typically defining the interactions between a role (known in the Unified Modeling Language as an actor) and a system, to achieve a goal.

Appendix 2 List of Figures

Appendix 3 List of Tables